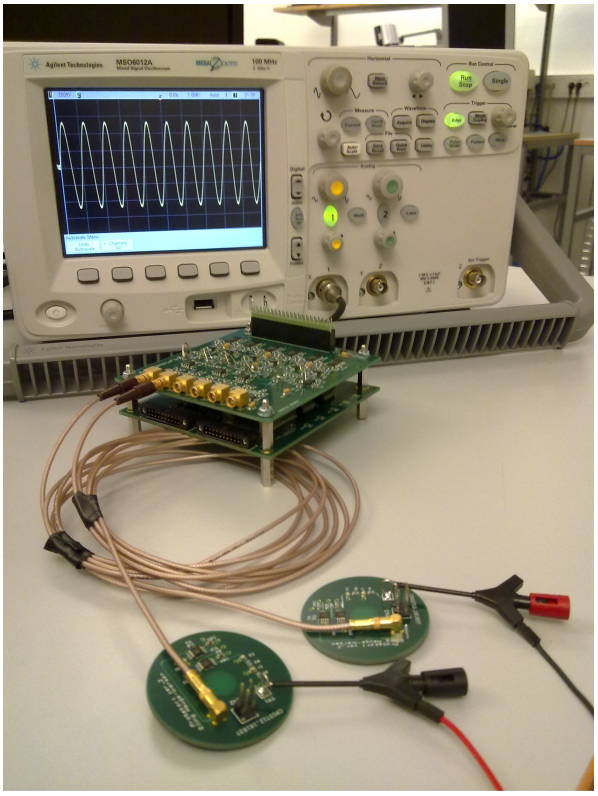
UNIVERSITETET I OSLO
Fysisk Institutt

E-feltinstrument for sonderakett



Masteroppgave

Elling
Hauge-Iversen

17. desember 2012

Sammendrag

Denne oppgaven omhandler utviklingen av et e-feltinstrument ment for målinger av elektriske felter i ionosfæren med en sonderakett som måleplattform. Instrumentet er en videreutvikling av et tilsvarende instrument utviklet ved rom og plasmagruppen til Universitetet i Oslo i 2007. Det har vært ønskelig å kunne gjøre AC- og DC-målinger på den samme kanalen og også kunne velge mellom differensielle eller single-ended målinger, noe som ikke har vært mulig tidligere. Oppgaven presenterer teoretisk bakgrunn, måleprinsipp, instrumentdesign og test- og kalibreringsresultater. Instrumentet er bygget for å kunne gjøre differensielle eller single-ended målinger med seks måleprober montert i nesepartiet på en sonderakett. Hver probe er på 40 mm i diameter og har en innebygd forforsterker. Instrumentets hoveddel er basert på formfaktoren til PC/104-standaren og består av to elektronikkort stablet oppå hverandre. Instrumentet mottar de forsterkede målesignalene fra forforsterkerne, filtrerer og digitaliserer ved hjelp av seks 24-bit Sigma-Delta analog-til-digitalomformere (ADC) og sender dem til telemetrienkoderen som kommuniserer med bakkestasjonen. Ved å gjøre differensielle målinger mellom to prober unngår man å bruke rakettkroppen som en flytende spenningsreferanse. Den funksjonelle testingen av systemet påviste et problem med den differensielle linjedriveren som overfører de digitale verdiene fra instrumentet til enkoderen. Problemet forsvant helt da driveren ble byttet ut med en eldre versjon, men det er ikke fastsatt hva som var årsaken til feilen i utgangspunktet. Dette er noe som må fastsettes i en revidering av instrumentet.

Takk til

Denne oppgaven ble til i perioden januar 2011 til desember 2012. Først og fremst må jeg takke mine to veiledere, førsteamanuensis Torfinn Lindem og stipendiat Tore André Bekkeng for vel utført veiledergjerning. Takk til Halvor Strøm på elektronikklaboratoriet for all hjelp med design og utlegg av kretskort. Takk til Senioringeniør Espen Trondsen for alt teknisk og praktisk knyttet til oppgaven. Takk til Andres, Bent, Cecilie og David på rom 303 for godt arbeids- og lærings-miljø. Takk til Anne og Hans i andre etasje for tant og fjas gjennom hele studietiden. Takk til Delonghi Magnifica for brunt flytende gull i sene timer. Sist og ikke minst må jeg takke Ragnhild, for å være den beste og tålmodigste kjæresten i verden.

Oslo, desember 2012
Elling Hauge-Iversen

Innhold

Forkortelser

AC	Alternating Current
ADC	Analog to Digital Converter
CMRR	Common Mode Rejection Ratio
DA	Digital til Analog
dB	desibel
DC	Direct Current
EISCAT	European Incoherent Scatter Scientific Association
ESR	EISCAT Svalbard Radar
EUV	Extreme Ultraviolet
FIFO	First In First Out
FIR	Finite impulse Response
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
HF	Høyfrekvent
IC	Integrated Circuit
ICI	Investigation of Cusp Irregularities
IMF	Interplanetary Magnetic Field
IO	Input Output
ISR	Incoherent Scatter Radar

JFET	Junction gate Field-Effect Transistor
LSB	Least Significant bit
Mbps	Megabit per second
MSB	Most Significant Bit
PCB	Printed Circuit Board
PCM	Pulse-Code Modulation
PLL	Phase-Locked Loop
PXI	PCI eXtensions for Instrumentation
RMS	Root Mean Square
SNR	Signal to Noise Ratio
UiO	Universitetet i Oslo
UV	Ultraviolet
VHDL	Very-high-speed integrated circuits Hardware Description Language

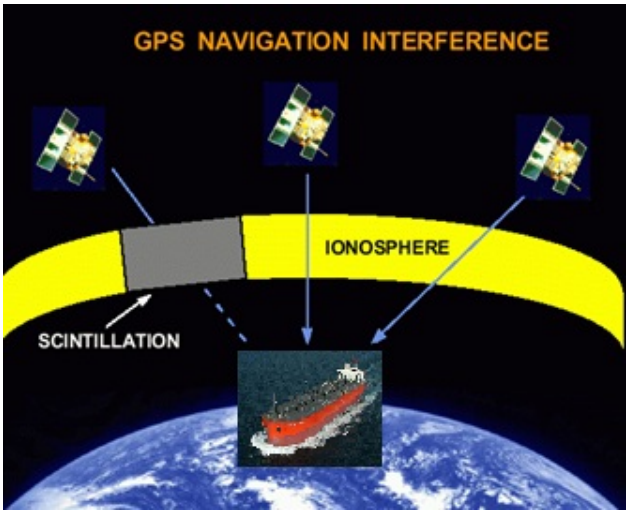
Figurer

Kapittel 1

Innledning

Denne oppgaven beskriver utviklingen og testingen av et system for måling av elektrisk felt (E-felt) i ionosfæren. Designet er en videreutvikling av et instrument fra 2007 utviklet ved universitetet i Oslo (Bekkeng 2007). Det nye systemet har seks målekanaler hvor vær kanal kan gjøre single-ended eller differensielle målinger. Systemet er konstruert for å gjøre målinger fra en sonderakett som skytes opp til 350 km høyde.

STAR er et sammarbeidsprosjekt mellom instituttene ved Matematisk og naturvitenskapelig fakultet ved UiO. Prosjektet har en ledende rolle i sonderakettprogrammet ICI som har som mål å øke forståelsen av turbulens og instabiliteter i ionosfæren. Når signaler fra satellitter går gjennom disse områdene av atmosfæren, kan slike instabiliteter være med på å forringe kvaliteten. Nøyaktigheten til satelittposisjoneringssystemer som GPS, Galileo og Glonass kan bli betydelig dårligere i perioder med stor aktivitet. Dette har negative konsekvenser for brukere med høye krav til posisjoneringsnøyaktighet. Virksomhet knyttet til olje- og gassleting trekker nordover ettersom polisen trekker seg tilbake. Nordøstpassasjen blir brukt mer og mer til skipstransport. Ved å øke kunnskapen om mekanismene bak disse atmosfæriske instabilitetene vil man også i større grad være i stand til å forutsi når forstyrrelsene kommer til å oppstå og også til en viss grad ha mulighet til å gjøre systemene mer robuste mot slike forstyrrelser.



Figur 1.1: Signaler fra satelitter kan bli forstyrret av turbulens i ionosfæren

1.1 Mål med oppgaven

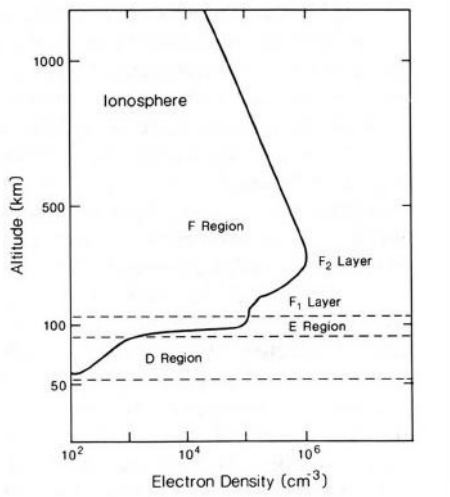
- Konstruere et instrument for målinger av elektriske felter i ionosfæren fra en sonderakett. Instrumentet skal ha mulighet for å gjøre målinger differensielt eller single-ended.
- Konstruere forforsterkere som kan plasseres inne i probekulene for å ha minst mulig signaltap.
- Instrumentets dimensjoner skal kunne legges på et eller flere kort etter formfaktoren til PC/104-standarden. På den måten kan man standardisere instrumentboksene som holder instrumentene ombord på raketten for lettere å integrere andre instrumenter uten å gjøre store mekaniske tilpasninger.
- Instrumentet skal kunne kobles direkte til rakettens enkoder, og fungere etter enkoderens spesifikasjoner.
- Det er også ønskelig å lage et PXI-basert test og kalibreringsystem for lettere å kunne kalibrere instrumentet når det er montert i raketten. Det bør legges til rette for at dette skal være mulig.

Kapittel 2

Elektriske feltmålinger

2.1 Ionosfæren

Ionosfæren strekker seg fra ca 60 til 500 km over jordoverflaten. Denne delen av atmosfæren blir dannet som følge av interaksjonen mellom solvinden, bestående av høyenergi elektromagnetisk stråling og ladde partikler, og gassmolekylene i atmosfæren. Gassmolekylene blir ionisert slik at det blir dannet plasma. Tettheten av frie elektroner er avhengig av forholdet mellom graden av eksitasjon og rekombinasjon. Figur 2.1 viser tettheten av frie elektroner som funksjon av høyden. Hovedsakelig kan ionosfæren inn i tre områder: D-, E- og F-laget. D-laget strekker seg fra 60 til 90 km høyde og har en forholdsvis lav tetthet av elektroner. Produksjonene av ioner og frie elektroner i D-laget skyldes UV-stråling som ioniserer nitrogenoksid NO. E-laget ligger mellom 90 og 150 km og produksjonen foregår hovedsakelig ved at røntgenstråling og ultrafiolett stråling ioniserer oksygen, O_2, og nitrogen, N_2. F-laget ligger i høyder over 150 km. Ioniseringen er her hovedsakelig drevet av ekstrem ultrafiolett stråling (EUV) som ioniserer N_2 og O. Tettheten av frie elektroner øker jevnt opp til ca. 250

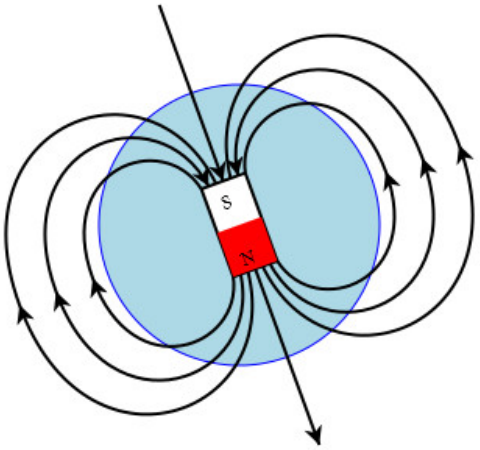


Figur 2.1: Elektrontetthet i ionosfæren som funksjon av høyde.

km, fordi partikkeltettheten avtar, slik at rekombinasjonstiden øker, samtidig som mindre av strålingen er absorbert. Over 250 km faller tettheten, selv om strålingen øker, fordi antallet partikler som kan ioniseres blir mindre. Graden av ionisering er sterkt avhengig av innstrålingen fra solen og varierer derfor gjennom døgnet. Det er også sesongvariasjoner som følge av jordens vinkling i forhold til solen. Solens solflekksyklus, med periode på ca 11 år er også påvirkende siden perioder med mye solflekker også er perioder med mye aktivitet.

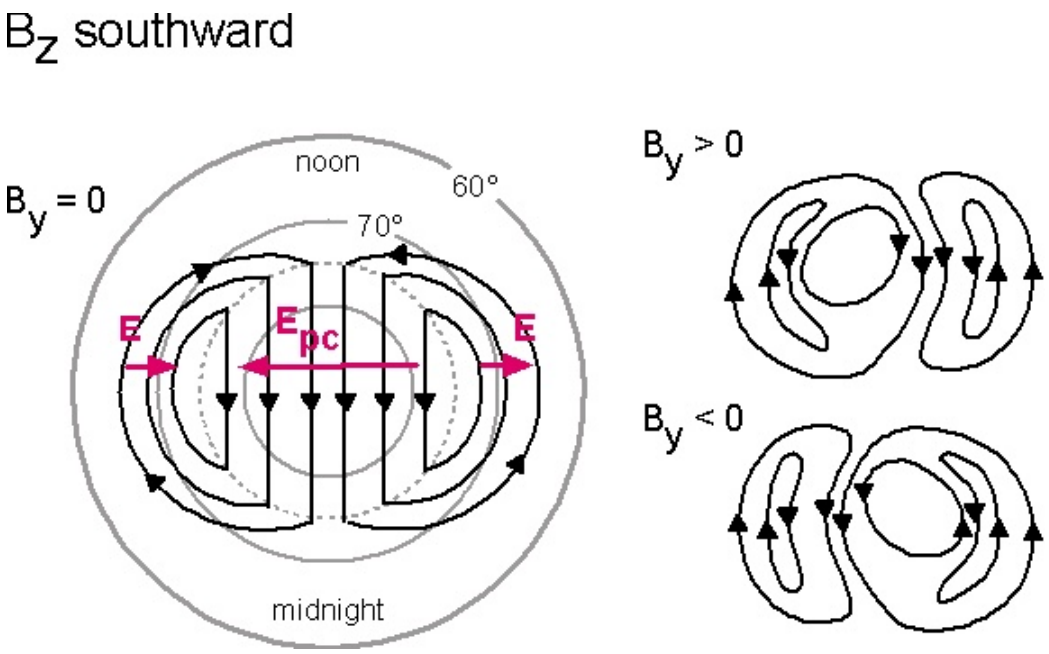
2.2 Elektriske felter i ionosfæren

Jordens magnetfelt kan sammenlignes med en magnetisk dipol på en stavmagnet tiltet 11° i forhold til den geografiske nord- og sydpol. Magnetfeltet bidrar til å beskytte jorden fra den strålingen som konstant kommer fra solen. Styrken på magnetfeltet ved jordoverflaten varierer fra ca. 30000 nT ved ekvator til ca. 60000 nT ved polområdene. Hvis man forflytter seg utover avtar feltet med ca. $\frac{1}{R^3}$, hvor R er radius fra jordens sentrum.



Figur 2.2: Jordens magnetfelt kan sammelignes med et dipolfelt fra en stavmagnet

Ved de magnetiske polene fungerer magnetfeltlinjene som en trakt. Ledningsevnen langs magnetfeltlinjene er veldig stor og derfor vil de elektriske feltene dannet av solvinden bli avbildet ned i øverste del av ionosfæren. Dette fører til en $E \times B$ drevet konveksjon i plasmaet. Figur 2.3 viser konveksjonen i ionosfæren drevet av det elektriske feltet. Ved å måle de elektriske feltene kan man få en indikasjon på størrelsen og retningen på strømmene.

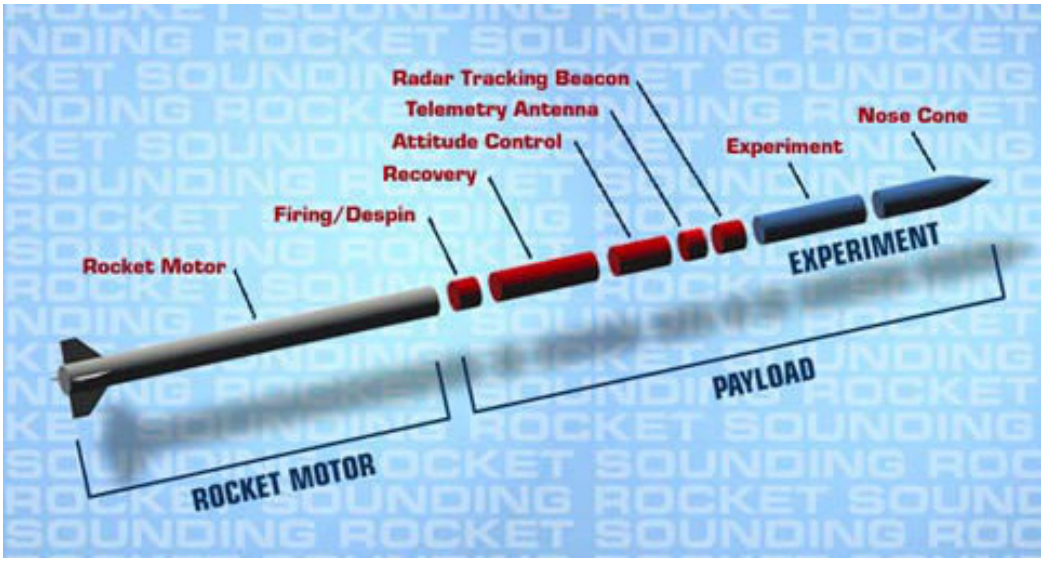


Figur 2.3: Konveksjon i ionosfæren som følge av elektriske felter avbildet ned fra solvinden når IMF er rettet sørover.

2.3 Sonderakett

Sonderaketter har blitt brukt til meteorologiske målinger i den øvre atmosfære siden sent på 50-tallet. En sonderakett kan som oftest deles inn i tre deler, én en- eller totrinns faststoffrakett, en servicemodul som tar seg av telemetri, ratekontroll, attityde o.l. og en nyttelast som kan ta et eller flere forskningsinstrumenter. Figur 2.4 viser en prinsipptegning av en sonderakett. Sonderaketter flyr i en parabelbane og er ikke kraftig nok til gå inn i en bane rundt jorden (Ceglia and Carey 2005). Grunnen til at sonderaketter brukes til målinger i ionosfæren er først og fremst at de er den billigste måten å gjøre direkte målinger. Forskningsballonger har en øvre høydebegrensning på ca. 40 km, mens lavbanesatellitter kan gå ned til ca. 160 km. Sonderakettene gjør det derfor mulig å gjøre direkte målinger i et område som man ellers bare kan gjøre indirekte observasjonsmålinger. Direkte målinger er også i mange tilfeller nødvendig for å få god nok romlig oppløsning. Radarmålinger med ISR (Incoherent Scatter Radar) er en av metodene for opservasjonsmålinger. EISCAT Svaldbard Radar (ESR) er en slik radar. ESR er plassert i Adventdalen nær Longyearbyen og sender på en frekvens på 500 MHz. Radaren har en grov rommlig oppløsning på 15×30 km, noe som gjør det vanskelig å få kunnskap ned på et ønskelig detaljnivå (Rinne et al. 2007). Fordelen ved

å ha en slik radar er at det gir mulighet til å overvåke de store bevegelsene i ionosfæren i sanntid og dermed kunne skyte når fenomenet man vil ha høyoppløselige målinger av oppstår.



Figur 2.4: Prinsipptegning av en sonderakett. *Foto: NASA*

2.4 Måling av elektriske felter fra sonderakett

Når raketten er i flukt vil det induseres et elektrisk felt når den passerer jordens magnetfelt.

$$\vec{E}_i = \vec{v} \times \vec{B} \tag{2.1}$$

Der $\vec{E}_i$ er det induserte e-feltet, $\vec{v}$ er hastighetsvektoren og $\vec{B}$ er magnetfeltet. Dette bidraget vil adderes til målingene av det faktiske e-feltet. slik at vi får

$$\vec{E}_r = \vec{E}' + \vec{E}_i = \vec{E}' + (\vec{v} \times \vec{B}) \tag{2.2}$$

Det $\vec{E}_r$ er det målte feltet mens $\vec{E}'$ er det faktiske efeltet som man er ute etter. Ligning 2.2 snus.

$$\vec{E}' = \vec{E}_r - \vec{E}_i \tag{2.3}$$

For at man skal kunne trekke fra det induserte e-feltet må rakettens attityde være kjent.

Hvis det antas at raketten beveger seg med en hastighet på $1000\frac{m}{s}$ vinkelrett på magnetfeltet som er på 50000nT vil det induserte feltet være

$$1000\frac{m}{s} \cdot 50000nT \approx 50\frac{mV}{m} \tag{2.4}$$

Dette er et stort tall med tanke på at elektriske felter i ionosfæren erfaringsmessig ligger på noen titalls mV/m (Pfaff et al. 2001), men kan komme opp i så mye som 150 mV/m (Maynard 2001).

2.5 Probeteori

Her følger noen betraktninger rundt probemålinger i et plasma, som er relevante når det konstrueres et målesystem.

2.5.1 Plasmateori

Debyelengde

En viktig plasmaparameter er Debyelengde. Fysisk er det den avstanden hvor en ladet partikkel influerer på omkringliggende partikler. Debyelengde er gitt som

$$\lambda_D = \sqrt{\frac{\varepsilon_0 k T_e}{ne^2}} \tag{2.5}$$

der ε_0 er permittiviteten i vakum, k er Boltzman's konstant, T_e er elektrontemperaturen n er elektrontettheten og e er elementærladningen.

En probe i et plasma vil opparbeide et negativt potensiale fordi elektronene har mye høyere mobilitet enn ionene. Dermed vil området rundt proben få et høyere antall positive ioner for å balansere. Tykkelsen på dette området vil være avhengig av probens potensial. Et slikt område rundt proben kalles en debyekappe.

2.5.2 Probe-plasmakobling

Potensialet til en probe i forhold til et omkringliggende plasma er gitt av strømbalansen

$$I_e + I_i + I_{ph} = I \tag{2.6}$$

hvor I_e er strømmen av elektroner fra plasmaet til proben, I_i er strømmen fra ionene i plasmaet og I_{ph} er strømmen pga. fotoemisjon av elektroner fra proben pga. innstråling.

Ifølge Langmuir (Mott-Smith and Langmuir 1926) er elektron- og ionestrømmen gitt som

$$I_e = -4\pi r^2 ne\sqrt{\frac{kT_e}{2\pi m_e}}\exp\left(\frac{eV}{kT_e}\right) \tag{2.7}$$

$$I_i = 4\pi r^2 ne\sqrt{\frac{kT_i}{2\pi m_i}}\left(1 - \frac{a^2 - r^2}{a^2}\exp\left(-\frac{r^2}{a^2 - r^2}\frac{eV}{kT_i}\right)\right) \tag{2.8}$$

Der r er proberadius, a er plasmakappens ytre radius, n er elektrontettheten i plasmaet når den er uforstyrret av probene, e er elementærladningen, k Boltzmanns konstant, m_i,m_e, T_i, T_e er masse og temperatur for elektroner og ioner og V er probepotensialet i forhold til plasmaet. Tykkelsen til plasmakappen er gitt ved:

$$a - r \approx \lambda_D\sqrt{-\frac{eV}{kT_e}} \tag{2.9}$$

Hvis proberadiusen er mye lengre enn kappetykkelsen kan ionestrømmen skrives som

$$I_i \approx 4\pi r^2 ne\sqrt{\frac{kT_i}{2\pi m_i}} \tag{2.10}$$

Hvis proben er i bevegelse sier Fahleson (Fahleson 1967) at en god tilnærming til ionestrømmen er

$$I_i \approx \pi r^2 ne\sqrt{\frac{8kT_i}{\pi m_i} + v^2} \tag{2.11}$$

Ligning 2.11 indikerer at ionestrømmen øker med probehastigheten. Det samme gjelder ikke for elektronstrømmen fordi probehastigheten er neglisjerbar i forhold til elektronhastigheten.

Strømmen som følge av fotoemmisjon kan skrives som

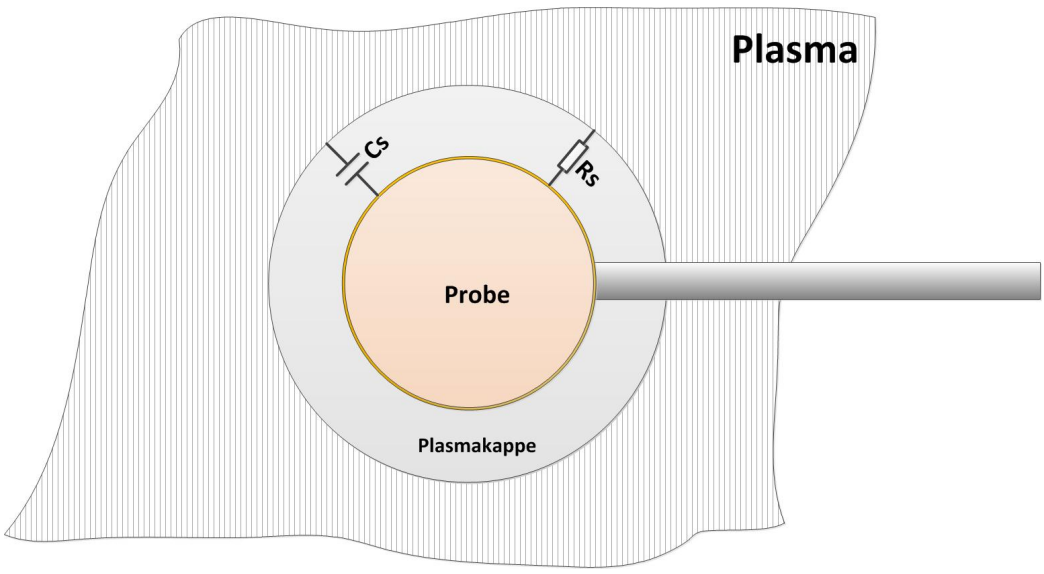
$$I_{ph} = \pi r^2 i_{ph} \tag{2.12}$$

Fotoemisjonen er sterkt avhengig av strålingen som treffer proben.

Ved å sette ligning 2.7 og 2.11 og 2.12 inn i 2.6 finner vi probepotensialet til å være

$$V = -\frac{kT_e}{e}\ln\frac{\sqrt{\frac{kT_e}{2\pi m_e}}}{\sqrt{\frac{kT_i}{2\pi m_i} + \frac{v^2}{16}} + \frac{i_{ph}}{4ne} - \frac{I}{4\pi r^2 ne}} \tag{2.13}$$

I et plasma med høy tetthet vil elektronstrømmen og ionestrømmen være dominerende. Siden elektronene har høyere mobilitet enn ionene, vil disse



Figur 2.5: Probe i plasma

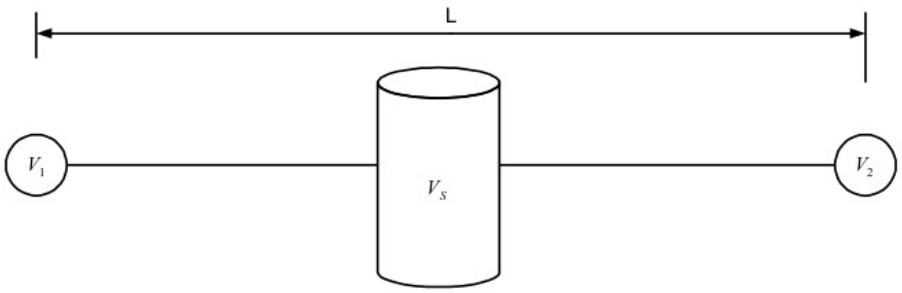
strømmene balansere proben på et negativt flytende potensial. Et teoretisk uttrykk for probens flytende potensiale er gitt når man setter I til 0 i ligning 2.13, men observasjoner har vist at de faktiske verdiene kan avvike fra denne. Stor fluks av elektroner og ioner med høy energi kan endre potensialet betraktelig. Det er også klare indikasjoner på at probens utforming har innvirknining på potensialet. For at man skal få så nøyaktige målinger som mulig, er det derfor viktig at probene er så identisk utformet som mulig(Fahleson 1967).

Plasmapotensialet er koblet til proben gjennom en kappemotstand R_s (Pedersen et al. 1998) og en kappekapasitans C_s. En probe i ionosfæren vil ha en kappemotstand på $10^5 - 10^6$ Ohm og en kappekapasitans på 5-10 pF (Bekkeng 2007).

2.5.3 Dobbeltprobeprinsippet

En veletablert metode for å måle elektriske felter er ved å måle potensialforskjellen mellom to prober. Ved å måle på denne måten unngår man å bruke rakettkroppen som en spenningsreferanse, noe som kan være gunstig siden rakettkroppen vil på samme måte som antenneprobene være utsatt for potensialendringer.

Elektrisk potensial er uttrykt som



Figur 2.6: Prinsippfigur for dobbeltprobekonfigurasjon.

$$V = \int \vec{E} \cdot \vec{dl} \tag{2.14}$$

Hvis det elektriske feltet er konstant mellom probene kan man omskrive det som

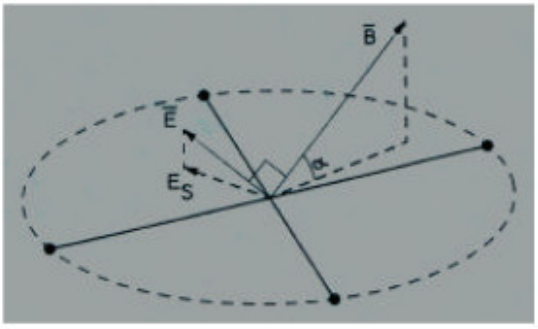
$$V = E \cdot L \tag{2.15}$$

Med to prober separert med en avstand L kan dette da gjøres om til

$$E_s = \frac{V_1 - V_2}{L} \tag{2.16}$$

Hvor V_1 og V_2 er potensialet til hver av probene, mens L er avstanden mellom disse. E_s er spinnplankomponenten til e-feltet, men vil være tilnærmet lik det virkelige feltet.

Et enkelt sett med dobbeltprober montert på nesen til en rakett vil kunne gjøre målinger for hver halve runde raketten roterer. For å få høyere oppløsning enn dette er man nødt til å bruke flere sett med prober montert som vist i figur 2.7.



Figur 2.7: Prinsippfigur for dobbeltprobekonfigurasjon med to kryssede dobbeltprober

2.6 Støy

Ethvert elektrisk signal vil ha en støykomponent. Støy kan ikke fjernes helt, men med god konstruksjon av kretser og med og skjerming er det mulig å dempe den betraktelig. Det er to typer støy som primært oppstår i probene når de beveger seg i et plasma. Den første er termisk støy, også kalt Nyquist støy, som genereres av kapperesistansen. Den andre er haglstøy som genereres når ladede partikler kolliderer med overflaten på probene (Gumett 1998).

2.6.1 Støytyper

Termisk støy

Termisk støy også kalt Johnson-Nyquist støy blir generert på grunn av ladningenes termiske bevegelse inne i en leder. Elektronene får en tilfeldig bevegelse hvor utslaget er proporsjonalt med temperaturen. Ved absolutt 0 vil derfor termisk støy være fraværende. Termisk støy sees ofte på som hvit støy, da den teoretisk er konstant over hele frekvensspekteret. For frekvenser under 100 MHz kan termisk støy kalkuleres ved å bruke Nyquists relasjon (Carter and Mancini 2002)

$$E_{th} = \sqrt{4kTR\Delta f} \tag{2.17}$$

eller

$$I_{th} = \sqrt{\frac{4kT\Delta f}{R}} \tag{2.18}$$

Hvor E_{th} er termisk støy i volt rms, I_{th} er termisk støy i ampere rms, k er boltzmann konstant, T er absolutt temperatur, R er resistans i Ohm og Δf er støyens båndbredde.

Haglstøy (eng: shot noise)

Haglstøy er et resultat fluktuasjoner i den elektriske strømmen som oppstår fordi elektronene i strømmen blir overført på diskrete tidspunkter. Når elektronene møter en potensialbarriere må de bygge opp nok potensiell energi før de kan passere. Når elektronet har nok energi vil den potensielle energien bli gjort om til kinetisk energi. Hvert individuelle elektron vil passere denne barrieren på et tilfeldig tidspunkt. Når dette skjer vil den frigjordte energien være et bidrag til haglstøyen. Haglstøy er omvendt proposjonal av strømmen i en leder. Rms strømmen for haglstøy kan skrives som (Carter and Mancini 2002)

$$I_{sh} = \sqrt{(2eI_{dc} + 4eI_0)\Delta f} \tag{2.19}$$

Hvor e er elementærladningen, I_{dc} er gjennomsnittlig dc strøm, I_0 er metningsstrømmen og Δf er båndbredden.

1/F-støy, Flekkstøy

Flekkstøy er omvendt proporsjonal med frekvens, derfor 1/F. Den opptrer i alle aktive og mange passive komponenter. Flekkstøy er proporsjonal med dc-strøm og kan derfor bli spist opp av den termiske støyen om den blir holdt lavt nok. Strømstøyen er gitt som (Carter and Mancini 2002)

$$I_n = K_i \sqrt{\ln \frac{f_{max}}{f_{min}}} \tag{2.20}$$

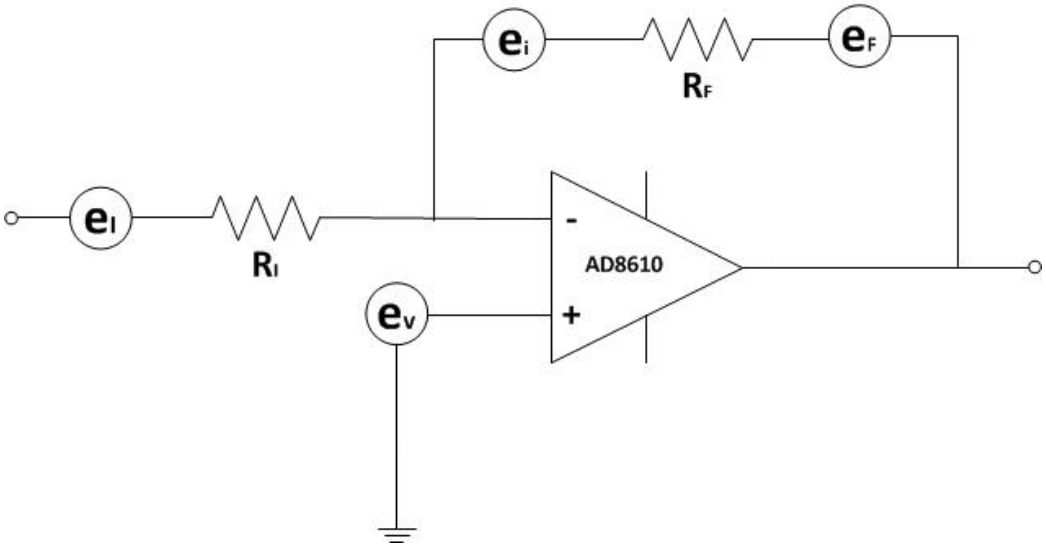
Hvor K_i er en proposjonalitetskonstant som representerer I_n ved 1 Hz. f_{max} og f_{min} er maximum og minimum frekvens i Hz.

2.6.2 Støy i plasma-probekobling

Haglstøy på en måleprobe bestemmes ut i fra kollisjonsraten mellom partikler og proben. Hvor mye haglstøy som blir generert har direkte sammenheng til størrelsen på probens overflate. Til tross for dette, er det ønskelig å lage kulene så store som mulig, siden store kuler har mindre impedans, noe som har en langt større fordel fordi sensorens impedans må være mindre enn impedansen til elektronikken for at ikke signalet skal mistes i kappemotstanden.

2.6.3 Støyanalyse av en inverterende forsterker

Figur 2.8 viser støykildene til en inverterende forsterker hvor e_I og e_F er den termiske støyen generert av motstandene R_I og R_F, e_i er operasjonsforsterkerens strømstøy multiplisert med tilbakekoblingsmotstanden R_F og e_V



Figur 2.8: Støyekvivalenskrets for en inverterende forsterker

er operasjonsforsterkerens spenningsstøy. Hvis man antar at støykildene er ukorrelerte kan man skrive den totale støyen i systemet som

$$e_{tot} = \sqrt{e_I^2 + e_F^2 + e_i^2 + e_v^2} \tag{2.21}$$

Operasjonsforsterkeren AD8610 er en høypresisjons JFET-forsterker med følgende støytall

- Strømstøytetthet: 5 fA/$\sqrt{Hz}$
- Spenningstøytetthet: 6 nV/$\sqrt{Hz}$

Hvis vi antar at $R_I = 100K\Omega$, $R_F = 300K\Omega$, båndbredden ΔF er 4KHz og temperaturen er 20°C (293,1 K).

$$e_I = \frac{R_F}{R_I}\sqrt{4kT\Delta F R_I} = 7.4\mu V \tag{2.22}$$

$$e_F = \sqrt{4kT\Delta F R_F} = 4.2\mu V \tag{2.23}$$

$$e_V = 6nV/\sqrt{Hz}\sqrt{\Delta F} = 0.37\mu V \tag{2.24}$$

$$e_i = 5\frac{fA}{\sqrt{Hz}}\sqrt{\Delta F} R_F = 0.9\mu V \tag{2.25}$$

Bruker vi ligning 2.21 får vi

$$e_{tot} = 8.5\mu V \tag{2.26}$$

Signal til støyratio (SNR) for en inverterende forsterker er gitt som

$$SNR = \frac{\frac{R_F}{R_I}V_{in}}{e_{tot}} \tag{2.27}$$

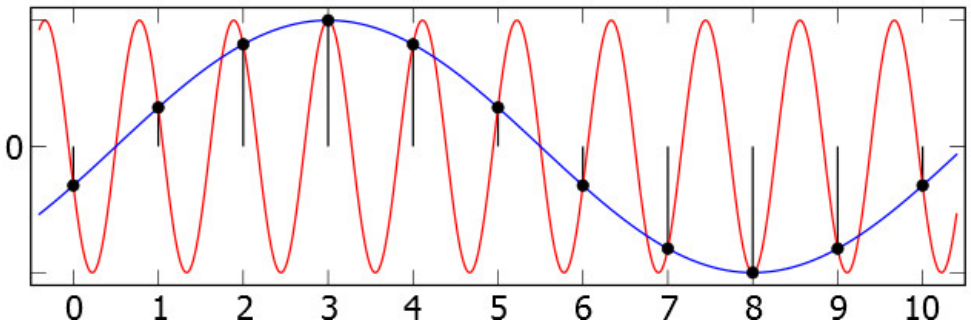
Hvis vi gjør den antagelse at signalet bør være ti ganger sterkere enn støyen for å være detekterbar får vi

$$\frac{\frac{R_F}{R_I}V_{in}}{e_{tot}} = 10 \rightarrow V_{in} = 10\frac{e_{tot}R_I}{R_F} \tag{2.28}$$

Hvis vi antar at $e_{tot} = 8.5\mu V$, $R_I = 100K\Omega$ og $R_F = 300K\Omega$ gir utregningen at det laveste detekterbare signalet ligger på $28\mu V$.

2.7 Datasampling

En analog-til-digital-konverter (ADC) transformerer analoge data, som regel i form av spenning, til en binær representasjon. Denne konverteringen skjer på diskrete tidspunkter. For at man skal kunne gjenskape det analoge signalet ut i fra de digitale verdiene sier Nyquists samplingsteorem at samplingsfrekvensen må være minst 2 ganger høyeste signalfrekvens, også kalt Nyquistfrekvensen. Når et signal har høyere frekvens enn halve samplingsfrekvensen, vil rekonstruksjonen av de diskrete samplingene gi et falskt signal med lavere frekvens enn originalen. Dette fenomenet kalles aliasing og er svært lite gunstig for måleresultatet.



Figur 2.9: Ved aliasing vil rekonstruksjonen av en samplet kurve gi en falsk kurve med lavere frekvens.

Figur 2.9 viser hvordan et signal med frekvens over nyquistfrekvensen kan bli avbildet galt. For å være sikker på å unngå aliasing må man sørge for at det analoge signalet blir filtrert før det konverteres. For å være sikker på at et

signal over Nyquistfrekvensen ikke bidrar til aliasing må den dempes så mye at den blir spist opp av støygulvet. For en AD-konverter kan man beregne SNR som (Baker 2007):

$$SNR = 6.02N + 1.76 \quad dB \tag{2.29}$$

Der N er antallet bit i konverteren som ligger over støygulvet. En konverter som har 16 bit over støygulvet blir da SNR:

$$6.02 \times 16 + 1.76 = 98 \quad dB \tag{2.30}$$

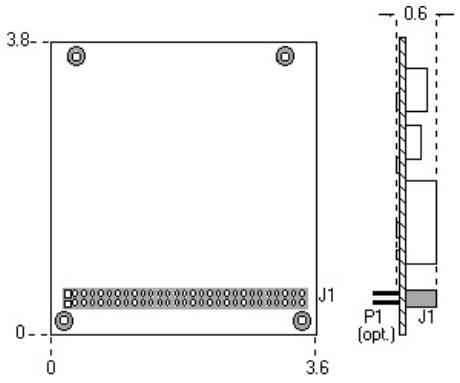
Signaler over Nyquistfrekvensen må da dempes minst 98 dB for at de ikke skal bidra til aliasing.

Kapittel 3

Spesifikasjoner

3.1 PC/104

Ved å lage kretskortene til alle instrumententene etter en standardstørrelse, vil det være mye enklere å inkludere nye instrumenter på raketten uten at det krever store mekaniske tilpasninger. Standarden som er valgt til dette formålet er basert på PC/104. Denne standarden opererer med kretskort på ca. størrelse med en diskett (90.17 × 95.89 mm) med en høydebegrensning på 15.24 mm. Standarden er konstruert på en slik måte, at det skal være mulig å stable flere kort oppå hverandre og koble de sammen med en kort-til-kort kontakt. Inn- og utgangkontakter er plassert på samme side av kortet slik at det vil det være enkelt å kunne bytte instrumentet uten å måtte endre den mekaniske utforming på instrumentboksen.



Figur 3.1: Dimensjoner for PC/104. Verdiene er i tommer. Bildet er hentet fra *www.pc104.org/pc104_specs.php*

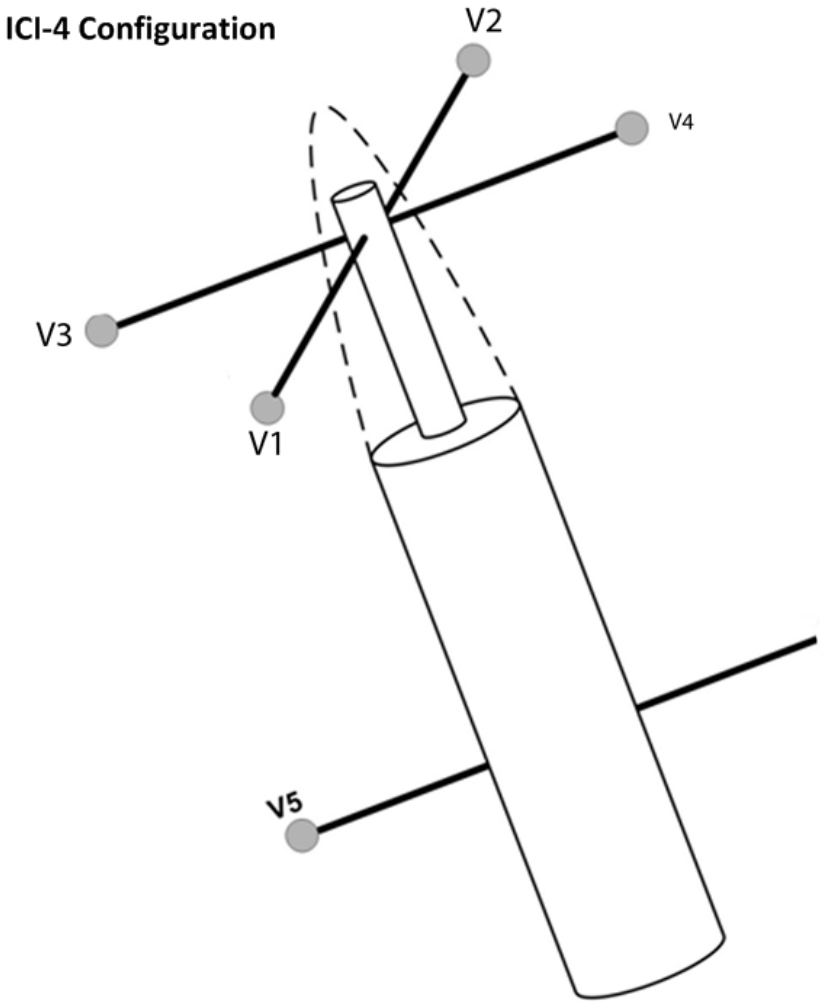
3.2 Probestørrelse

Selve proben er en kule på 40 mm i diameter. Den er laget i aluminium og belagt med titaniumnitritt.

3.3 Probespesifikasjoner

For ICI-4 er det aktuelt med 5 måleprober. Figur 3.2 viser probeoppsettet på raketten. 4 prober vil sitte på instrumentbommer i nesepartiet på raketten og den siste vil være montert på en bom på siden av raketten. Under utskytning vil bommene være innfelt og beskyttet inne i raketten. Etter ca. 60 sekunder etter oppskytning vil dekslene bli kastet av og bommene utfelt. E-feltinstrumentet er satt opp med seks målekanaler med følgende oppsett:

Kanal 1	Singel V5
Kanal 2	Differentiell V3 - V5
Kanal 3	Singel V3
Kanal 4	Differentiell V4 - V3
Kanal 5	Singel V4
Kanal 6	Differentiell V1 - V2

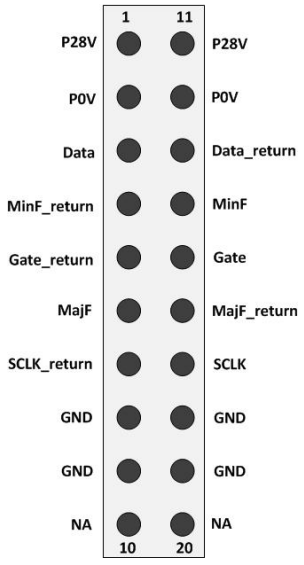


Figur 3.2: Plassering av probene på raketten

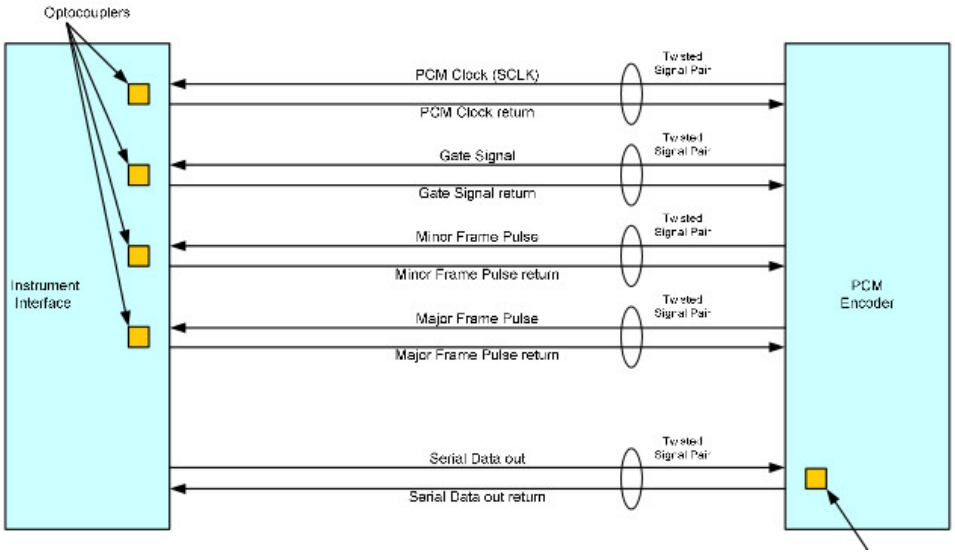
3.4 Grensesnitt mot enkoderen

Grensesnitt		
Pin nr.	**Navn**	**Beskrivelse**
1 og 11	P28V	28 volt input fra enkoder
2 og 12	P0V	Enkoders referansenivå
3	Data	Signal ut. Data fra instrument til enkoder
4	MinF_return	Invertert av MinF
5	Gate_return	Invertert av Gate
6	MajF	Major Frame. Signal inn. Signal fra enkoder som merker start på et nytt format.
7	SCLK_return	Invertert av SCLK
8,9,18,19	GND	Instrumentjord
10 og 20	NA	Ikke tilkoblet
13	Data_return	Invertert av Data
14	MinF	Minor Frame. Signal inn. Signal fra enkoder som signaliserer start på en ny ramme
15	Gate	Signal inn. Signaliserer når databit kan leses ut på datalinjen
16	majF_return	Invertert av MajF
17	SCLK	Signal inn. Klokke fra enkoder på 3.33 MHz

Figur 3.3 viser grensesnittet instrumentet skal kobles til. Instrumentet får 28 Volt samt data-, klokke- og styringssignaler gjennom en enkelt kontakt. Alle signaler mellom instrument og dekoder overføres differensielt for å gjøre overføringen mer robust for støy. De differensielle signalene mottas av en optokopler på mottakersiden for å unngå jordsløyfer. Utgangsimpedansen til driveren bør matches til kabelimpedansen som ligger på ca 100 Ω. Figur 3.4 viser hvordan tilkoblingen mellom enkoder og instrument bør være.



Figur 3.3: Instrumentets grensesnitt mot enkoder

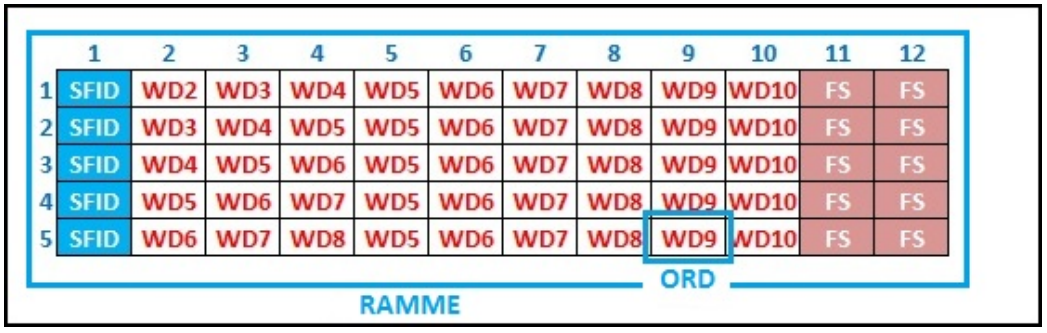


Figur 3.4: Instrument til encodertilkobling. De gule kvadratene symboliserer optokoblere. Bilde hentet fra(ARR Payload Services 2009)

3.5 Telemetriformatet

Data fra raketten ned til bakkesta-
sjonen går gjennom en datakanal som deles mellom alle instrumentene ombord. Enkoderen som sitter i raketten har som oppgave å lese av instrumentene på raketten, og tidsmultiplekse måleverdiene i en rekkefølge som er bestemt på forhånd. For at mottakeren skal kunne vite hvilke måleverdier som kommer når må datastrømmen også inneholde informasjon om dette i form av synkroniseringsord. Plasseringen av synkroniseringsord, måleverdier og telleverdier er fastsatt av PCM-formatet.

Datastrømmen er delt opp i format, ramme ord og bit. Et format består av 64 rammer, en ramme består av 72 ord og et ord består av 16 bit. Hvert ord innenfor en ramme har en spesifikk betydning. Figur 3.5 viser en prinsipiell oppbygging av en ramme med dataord, synkroniseringsord og tellere. Hvor mange ord i en ramme et instrument blir tildelt, har stort sett sammenheng med hvor ofte verdien fra dette instrumentet trenger å bli oppdatert. Kommutering er et forholdstall X:1 som sier noe om hvor mange ganger data fra et instrument opptrer i en ramme. Hvis et instrument opptrer en gang per ramme, kalles det for hovedkommutering. Hvis X>1, kalles det for superkommutering. Noen instrumenter trenger ikke å bli oppdatert for hver ramme. Dette kan f.eks. være tilfelle for en temperatursensor, hvor endringen fra ramme til ramme er minimal. Derfor kan man da bruke en plass i rammen



Figur 3.5: En ramme består av underrammer som igjen består av ramme-id, dataord og synkroniseringsord. (ARR Payload Services 2009)

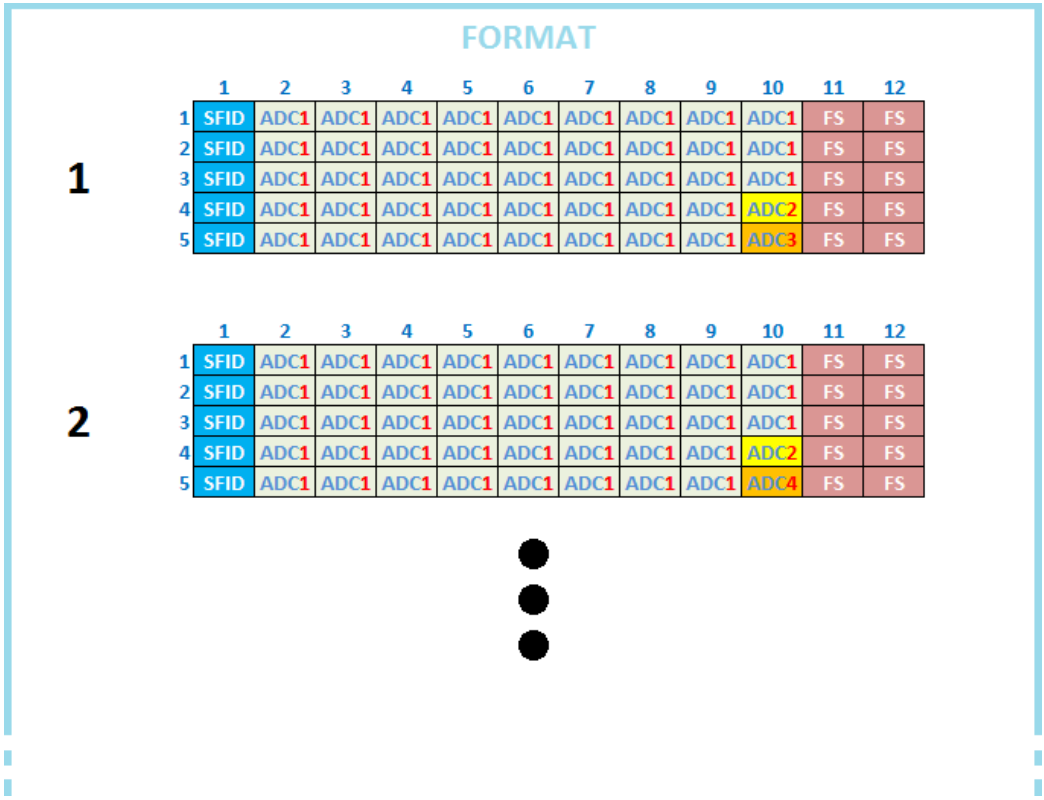
til å overføre data fra forskjellige instrumenter ut i fra hvilken ramme man befinner seg i. Dette kalles for subkommutering.

3.6 Dataoverføring fra instrument til enkoder

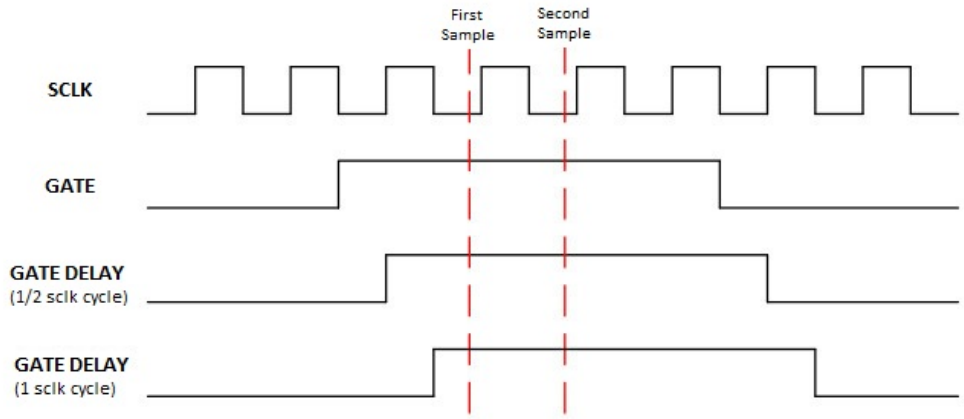
Dataoverføringen blir styrt av fire signaler fra enkoderen. Minor- og Major frame, indikerer starten på en ny ramme eller format. Disse brukes til å synkronisere tellerene i instrumentet med enkoderen. Når enkoderen er klar til å motta verdier fra instrumentet går gate høy. Siden instrumentet har flere kanaler er det viktig at det er synkronisert med enkoderen slik at den sender den riktige dataen.

Dataene overføres med MSB først. Dataene skiftes ut på stigende flanke til SCLK når gate er høy. På grunn av timing vil ikke enkoderen sample det første bitet før etter ca. 1 1/2-klokke syklus. Figur 3.7 viser hvordan man er nødt til å forsinke gate-signalet internt i instrumentet for ikke å miste MSB.

Hver kanal får tildelt 17361 Hz av telemetrilinken ned til bakken. Dette setter begrensning for samplingsfrekvensen.



Figur 3.6: Flere rammer i et format. Tre analog-til-digitalomformere deler datalinjen. ADC1 er et eksempel på superkommutering, ADC2 på hovedkommutering mens ADC3 og ADC4 deler deler det samme ordet i rammene og er derfor et eksempel på subkommutering



Figur 3.7: Figuren viser hvor enkoderen setter gate høy og hvor første sample leses. For at enkoderen ikke skal miste MSB må gate forsinkes. (ARR Payload Services 2009)

Kapittel 4

Design

Dette kapittelet fokuserer på designet av elektronikken til e-feltinstrumentet. Instrumentet består av fire forskjellige deler: probe, forforsterker, analogkort og digitalkort. Proben følger potensialet til plasmaet som forklart i kapittel 2. Forforsterkeren er plassert inne i probekulen. Den har som oppgave å filtrere og forsterke signalet før det sendes til analogkortet over en koaksialkabel. Analogkortet filtrerer og digitaliserer signalet. Det er i dette kortet hvor valgene om det skal brukes differensielle eller single ended signaler tas. Signalene digitaliseres av seks ADCer. Digitalkortet tar i mot de digitale signalene fra analogkortet. Hovedoppgaven til dette kortet består i å styre ADCene og å kommunisere med enkoderen som sitter i raketten. Det er også på digitalkortet at DC-DC-omformerene sitter. Disse forsyner instrumentet med 5, 2.5 og 1.2 volt.

4.1 Valg av komponenter

Når man skal velge komponenter til et design, er det tradisjonelt fire krav som må overveies.

- Tekniske krav
- Driftslevetid
- Pålitelighet
- Økonomi

De tekniske kravene går på ting som driftsspenninger, temperaturkrav, krav til frekvens, dimensjoner osv. I en sonderakett har man som regel begrenset fysisk plass å utfolde seg på og det er også begrenset hvor mye strøm som kan

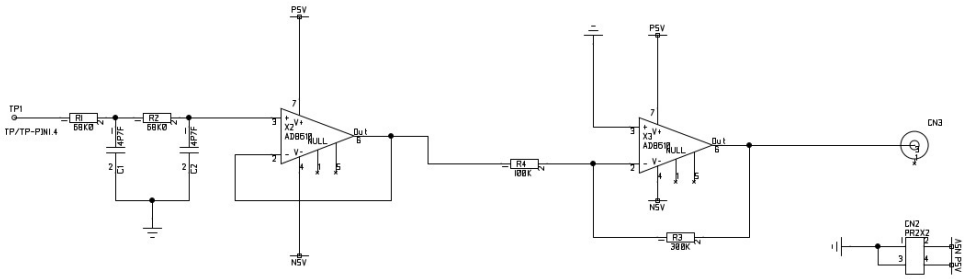
brukes. Konstruksjoner som skal fungere i omgivelser med mye stråling, har ofte krav opp mot dette. Driftslevetiden i en sonderakett overstiger sjelden en halvtime men siden man kun har en mulighet til å gjøre målingene er det viktig at påliteligheten til komponentene er høy. Økonomi kan også ha en innvirkning på valgene. Det er dumt å bruke for mye penger på en komponent bare for å presse ut litt ekstra ytelse.

4.2 Forforsterker

Probekortet er 40 mm i diameter og sitter inne i sensorkulene. Figur 4.1 viser det skjematiske oppsettet av forforsterkerdelen som består av tre deler:

- Et 2. ordens RC lavpassfilter med knekkfrekvens på 50 KHz for å filtrere bort forstyrrende signaler og støy utenfor det interessante området.
- En spenningsfølger for å unngå at signalet blir dempet som følge av spenningsdeling mellom den veldig høye plasmakappemotstanden og den resterende kretsen.
- En inverterende forsterker for å løfte signalet høyere opp fra støygulvet.

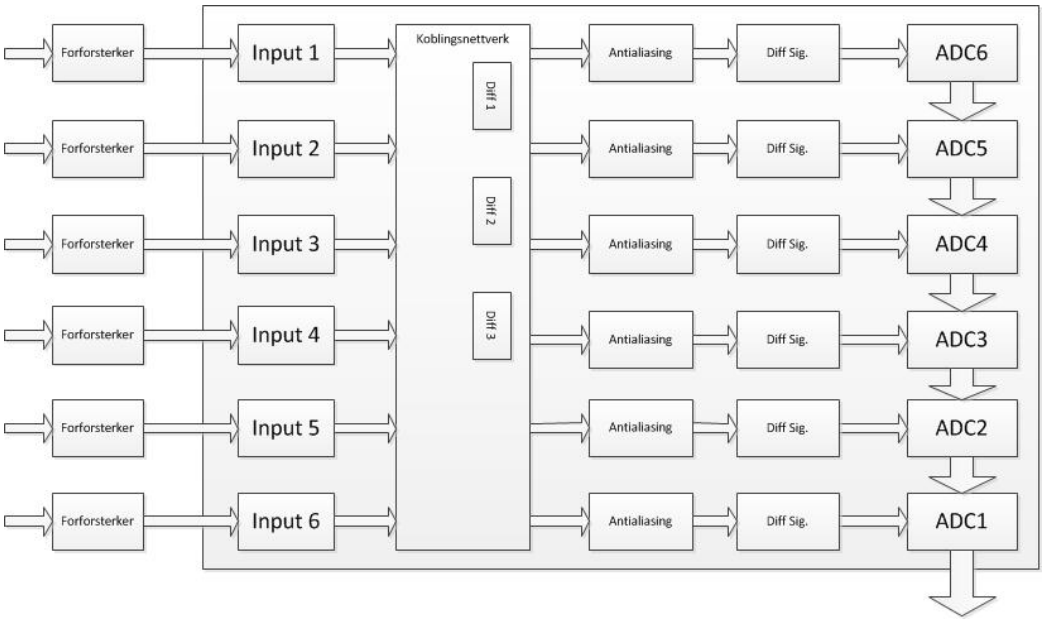
Operasjonsforsterkeren AD8610, er hovedsakelig valgt pga. den høye inngangsmotstanden som kreves for å unngå at signalet forsvinner som følge av spenningsdeling mellom kappemotstanden og inngangsmotstanden. Den har veldig lav inngangskapasitans (ca 15 pF), liten offset-spenning og genererer veldig lite støy ($6nV/\sqrt{Hz}$).



Figur 4.1: Skjematisk oppsett av probekortet.

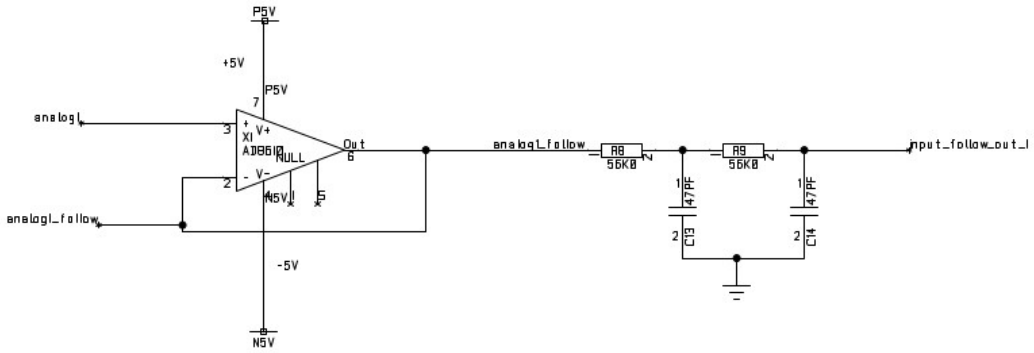
4.3 Analogkort

Analogkortet har seks signalinnganger. Det er konstruert slik at det skal kunne kobles oppå et digitalkort av tilsvarende dimensjoner. Signalene som kommer inn blir fordelt i et koblingsnettverk som bestemmer om signalene skal måles mot et annet inngangssignal differensielt, eller om det skal bare måles mot signaljord single-ended. Signalene går deretter gjennom et 8-pols lavpassfilter for å hindre aliasing når de digitaliseres. Før signalene går inn i AD-konverteren blir de gjort om til to differensielle signaler. Deretter går de differensielle signalene inn i ad-konverteren, blir digitalisert og sendt til digitalkortet.



Figur 4.2: Blokkskjema av den analoge elektronikken. Forforsterkerene er plassert inne i probekulene, mens den resterende elektronikken ligger på analogkortet. Inngangene fra probe 3 og 4 er byttet om for lettest mulig å kunne gjøre differansemålinger. Koblingsnettverket består av en rekke motstander på 0 Ω som fungerer som brytere. AD-konverterene er koblet i daisy-chain. Kontrollsignaler til ad-konverterene er utelatt for å gjøre figuren mer oversiktlig.

4.3.1 Spenningsfølger

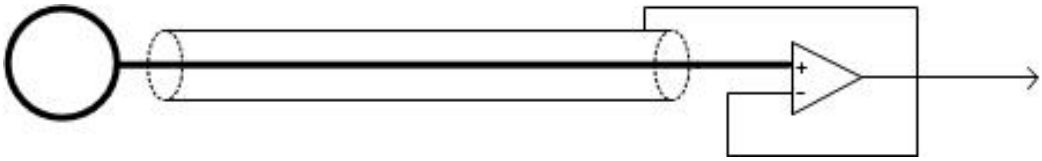


Figur 4.3: Ved å sette en spenningsfølger på inngangen, vil ikke analogkortet belaste probekortet.

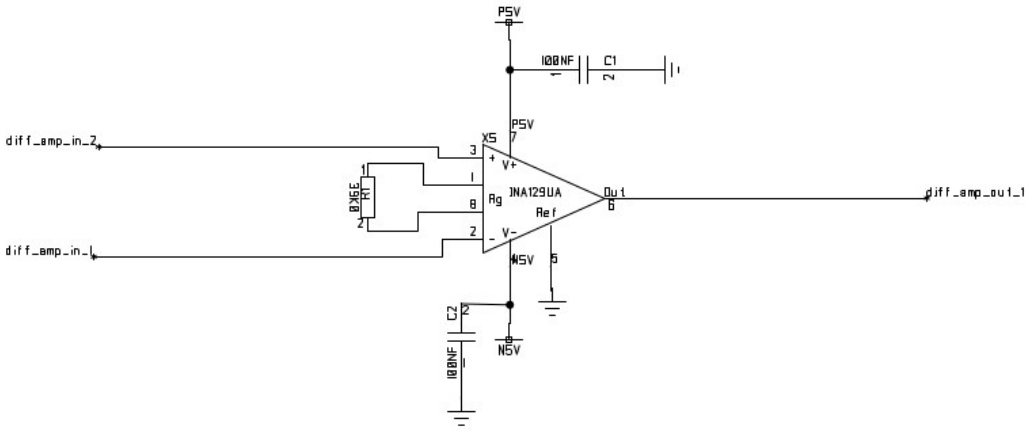
Figur 4.3 viser spenningsfølgeren på inngangen til analogkortet. Ved å konfigurere en operasjonsforsterker på denne måten oppnår man en høy inngangsimpedans og vil få et veldig lite spenningsfall gjennom kabelen fra probekortet da spenningsdelingen mellom motstanden i kabelen og inngangsimpedansen på opampen er veldig forskjøvet over på opampen. På samme måte gjør den lave utgangsimpedansen til opampen at signalspenningen ikke blir degradert som en følge av spenningsdeling mellom opampen og resten av den etterfølgende kretsen. Man unngår også at kretsen ikke trekke strøm fra proben, men fra opampen. En annen grunn til at det er benyttet en spenningsfølger er for å koble den lavohmige utgangen til ytterkappen på kabelen fra proben slik at kabelens kapasitans mellom kappen og senterlederen virtuelt sett blir mindre. Denne teknikken kalles bootstrapping og er illustrert i figur 4.4

4.3.2 Koblingsnettverk

Mellom spenningsfølgerene på inngangstrinnet og resten av kretsen er det satt inn et nettverk av 0 Ohmige motstander. Hensikten er å kunne velge mellom single-ended eller differensielle signaler ved å kunne lede signalene inn til eller rundt differanseforsterkerene. Hvilke signaler som skal sammenlignes differensielt må bestemmes på forhånd slik at man kan sette på de



Figur 4.4: Bootstrapping.



Figur 4.5: Differanseforsterkeren sammenligner to signaler og gir ut forskjellen $V_{in}^{+} - V_{in}^{-}$.

motstandene som gir kobling dit man vil at signalene skal gå.

4.3.3 Differanseforsterker

Som differanseforsterker er det valgt INA129. Den har en CMRR (Common Mode Rejection Ratio) på 95 dB, genererer lite støy og har lav offset- og biasstrøm. Den aksepterer også spenninger i et bredt område. Forsterkningen bestemmes med en enkelt motstand R_g. Matematisk kan denne uttrykkes som:

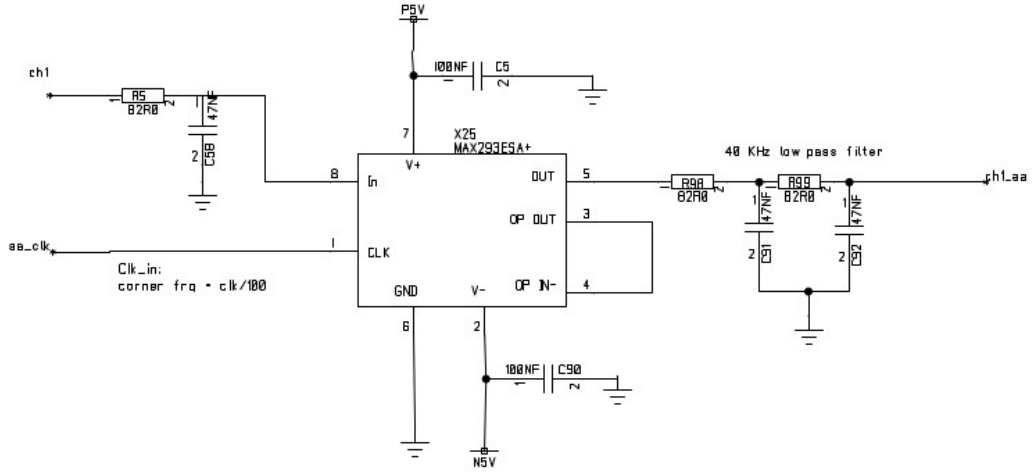
$$G = 1 + \frac{49.4\ k\Omega}{R_g} \tag{4.1}$$

Forsterkningen blir 1 når man lar være å koble til en motstand. Figur 4.5 viser differanseforsterkeren slik den er koblet på kortet.

4.3.4 Antialiasingfilter

For å unngå aliasing når signalene blir digitalisert settes det inn et MAX293 aktivt lavpassfilter med regulerbar knekkfrekvens. Knekkfrekvensen bestemmes av klokken som genereres i FPGAen på digitalkortet og er gitt ved:

$$f_k = \frac{CLK}{100} \tag{4.2}$$



Figur 4.6: For å unngå aliasing, brukes et aktivt lavpassfilter. Knekkfrekvensen en hundredel av klokkefrekvensen satt på inngang 1.

MAX293 er et 8.ordens elliptisk lavpassfilter med en dempning på 80 dB i stoppbåndet og en rippel på 0.15 dB i passbåndet. Offset-spenning kan variere inntil. ±150 mV fra IC til IC, men en enkelt IC vil ha konstant offset over hele båndbredden. Figur 4.6 og figur 4.7 viser frekvensresponsen til et 8. ordens elliptisk filter. Før og etter filteret er det satt inn et 1.- og et 2.-ordens lavpassfilter. Filteret før er der for å hindre at frekvenser høyere enn halve klokkefrekvensen slipper inn. Filteret etter skal hindre at signalet fra klokken skal smitte over på signallinjen og videre til AD-konverterene.

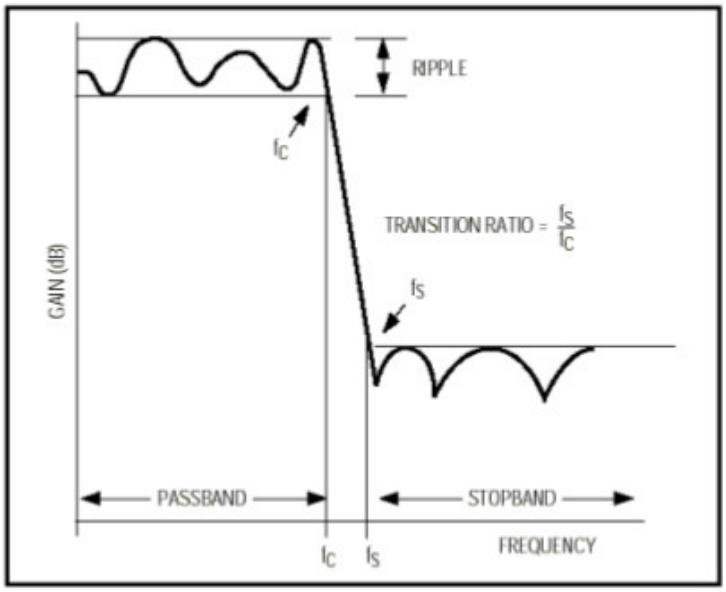
4.3.5 Analog til digitalomformer (ADC)

En Sigma-Delta digitalomformer er en veldig rask og nøyaktig type AD-konverter som benytter seg av pulsmodulasjon. Konverteren genererer pulser med en amplitude V og en lengde t. Avstanden mellom pulsene bestemmes av spenningen til det analoge signalet v. Pulsene blir generert ved at signalet integreres opp til man når tilstanden

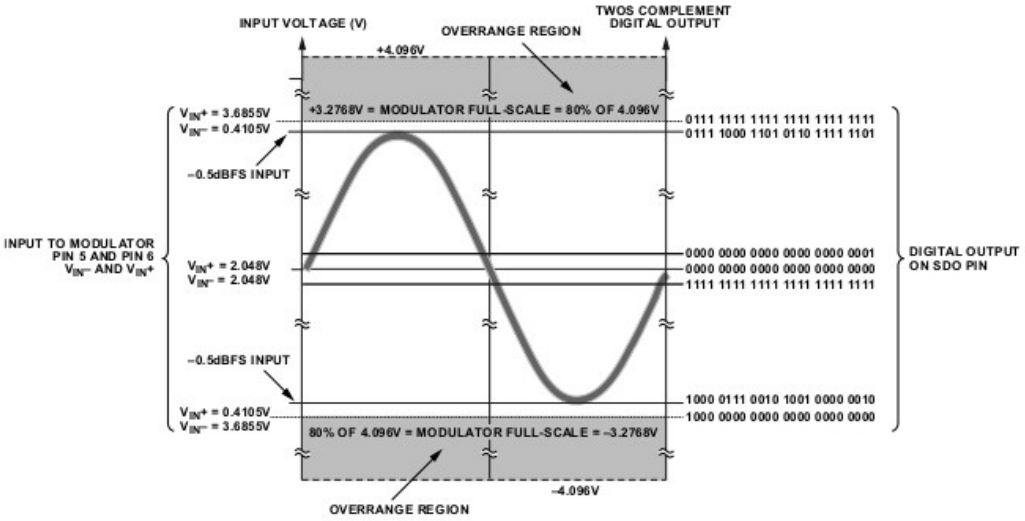
$$V \cdot t = \int vdt \tag{4.3}$$

Når dette skjer genereres det en puls og en ny integrasjon starter. Tiden mellom pulsene er derfor avhengig av spenningsnivået til inngangssignalet. Summen av antallet pulser i en gitt tidsperiode vil være den digitale representasjonen av den analoge verdien.

AD-konverteren som er valgt, AD7765, er en 24-bits Sigma-Delta konverter. Den kommer i en 28-pins TSSOP-pakke og har en konverteringsrate på



Figur 4.7: Frekvensenresponsen til MAX293, et 8. ordens elliptisk lavpassfilter. Bildet er hentet fra MAX293 datablad.



Figur 4.8: Signalets spenningsverdi mot konverterens digitale utgangsverdi. Bildet er hentet fra AD7765 datablad.

opp til 156 KHz. Flere konvertere kan kobles i daisy-chain, noe som forenkler designet betraktelig. Konverteren er avhengig av en ekstern klokke som blir generert i FPGAen på digitalkortet. Frekvensen til den eksterne klokka bestemmer samplingsraten, knekkfrekvenser på de innebygde filtrene og utlesningsraten. Konverteren har også et innebygd FIR-filter, som gir økt beskyttelse mot aliasing av signaler som ligger over Nyquist-frekvensen.

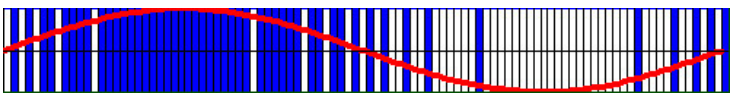
AD-konverteren AD7765 har en innebygd differanseforsterker. Det er anbefalt å drive signalene differensielt. Det er derfor plassert en ekstern opamp AD8021 foran for å konvertere til differensielle signaler. Opampen som brukes er spesialtilpasset til å drive signaler inn til AD-konvertere. Den genererer lite støy og har en lav drift i inngangsstrøm og -spenning. Figur 4.10 viser hvordan omformingstrinnet og komponentene er plassert rundt den differensielle forsterkeren.

Konverteren krever en 4.096 V referansespenning på pinnen V_{ref} fra en høypresisjonsreferanse. Siden konverteren opererer med differensielle signaler gir dette et signalspenn på 8.1V. Ca. 80% av dette spennet kan bli tilført inngangspinnene. Det vil si at et maksimum på $\pm$ 3.2768 V p-p kan påføres hver av inngangene. LSB representerer, når det er 24 bit oppløsning, en spenningsendring på $0,38\mu V$. I seksjon 2.6.3 fikk vi at et signal må være minst $28\mu V$ for å være detekterbart. Det vil si at de sju laveste bitene representerer støy og vil ikke være noe vits i å overføre. Figur 4.8 viser sammenhengen mellom inngangssignalet og de digitale utgangsverdiene til konverteren. Det er verdt å merke seg at verdiene ut kommer som toerkomplement.

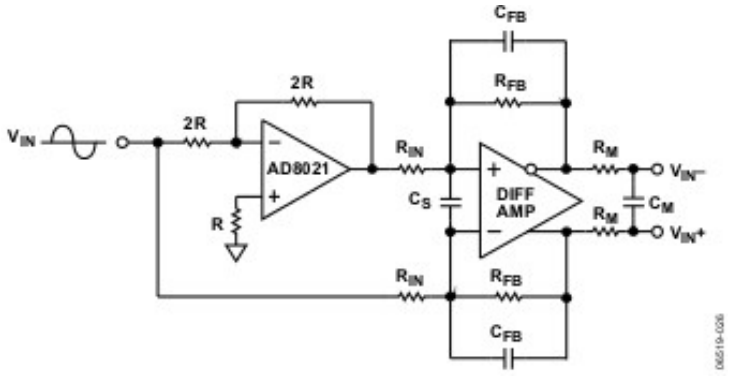
Konverteringsresultatene blir klokket ut med en frekvens som er halve av frekvensen på klokkeinngangen. Dataene blir klokket ut serielt samtidig som signalet $\overline{FSO}$ går lavt. Hvert bit av konverteringsresultatet blir klokket ut på stigende flanke og er gyldige på den fallende flanken. En overføring består av 24 databit, 5 statusbit og tre nuller, tilsammen 32 bit.

4.3.6 Daisy chaining

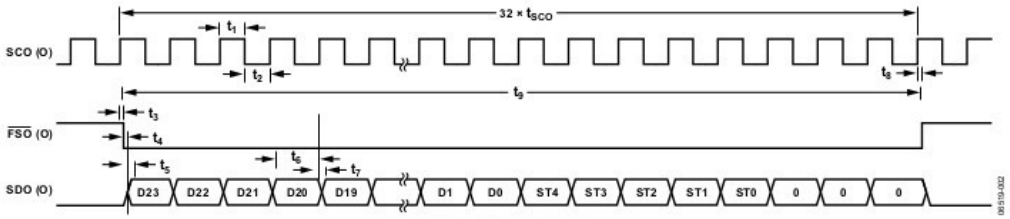
AD7765 støtter daisy chaining. Dette vil si at flere konvertere kan dele den samme interfacelinjen ved at konverterene er koblet i en kjede hvor en konverter kun kommuniserer med konverteren foran og bak. Ved å koble på denne



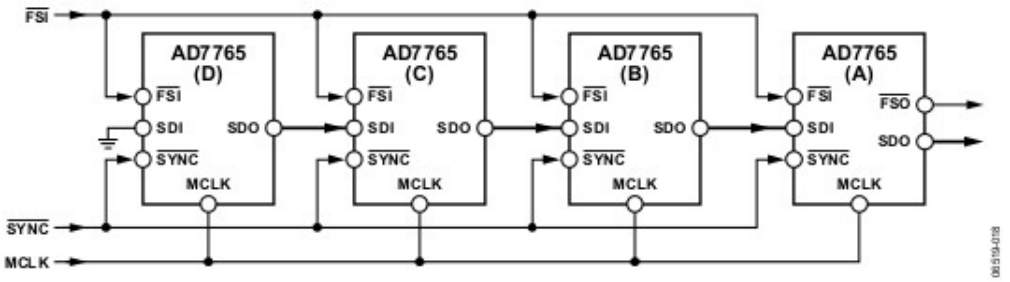
Figur 4.9: Pulsmodulasjon av et signal.



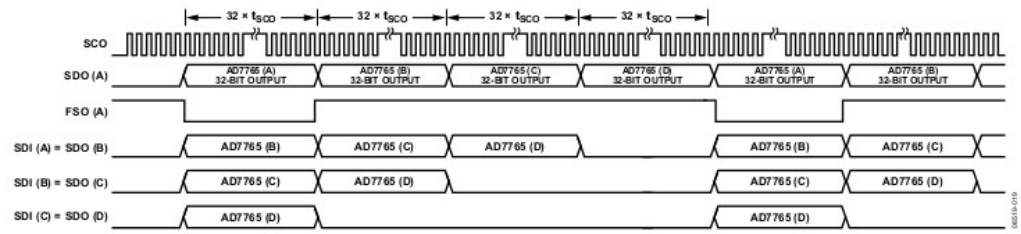
Figur 4.10: Omforming fra single-ended til differensielle signaler. Viser også hvordan komponentene er plassert rundt AD-konverterens innebygde diff-forsterker. Bildet er hentet fra AD7765 datablad.



Figur 4.11: Timing diagram for en lesesyklus fra AD7765.



Figur 4.12: AD7765 koblet i daisy chaining.



Figur 4.13: Timing diagram for en lesesyklus fra flere AD7765 koblet i daisy chaining.

måten vil man kunne redusere bruken av komponenter og også redusere antallet tilkoblinger som trengs. Figur 4.12 viser en prinsipiell kobling av fire konvertere. For at konverterne skal kunne operere på denne måten må de dele samme klokken og synkroniseringssignal. Figur 4.13 viser hvordan dataene blir klokket ut fra konverterene og inn i konverteren foran i kjeden. Denne prosessen foregår helt til den fremste konverteren har klokket gjennom samtlige verdier. Den fallende flanken av $\overline{FSO}$ signaliserer starten på en ny utlesning, og holder seg logisk lav i de første 32 klokkeperiodene og går deretter logisk høy resten av perioden.

4.4 Digitalkort

Digitalkortet har to oppgaver. Den ene er å forsyne med driftsspenninger på ±5volt, 2.5 volt og 1.2 volt. Den andre er å være bindeleddet mellom adkonverterene på analogkortet og enkoderen til raketten. All kommunikasjon gjøres med en 256-pins FPGA (Field Programmable Gate Array) fra Altera. Som nevnt i kapitter 3 må signalene transformeres til eller fra differensielle signaler med henholdsvis en differansedriver eller optokobler for å kunne kommunisere med enkoderen. Digitalkortet forsyner også probene med ±5 volt og jord.

4.4.1 Konnektorer

Til å forsyne probene med spenning og kommunisere med enkoderen brukes det to 20 pins Datamate-kontakter fra Harwin. Disse er brukt fordi de tar lite plass og holdes på plass av skruer slik at kontaktene ikke skal ristes løs av vibrasjonene fra motortrinnene fra raketten. Mellom analog- og digitalkortet kommuniseres det med en board-to-board kontakt fra Harwin med 20 pinner.

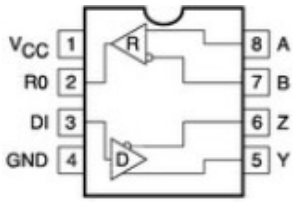
4.4.2 Spenningsforsyning

Digitalkortet har tre spenningsregulatorer. Fra enkoderen mottar kortet en forsyningspenning på 28 volt. Denne blir først omgjort til ±5 Volt i en dual output DC-DC konverter fra TDK-Lambda. Denne kan levere ±5 volt med en effekt opp til 24 watt. Videre er det to dc-dc konvertere av typen MCP1827 som konverterer 5 volt ned til 2.5 volt og 1.2 volt. Konverteren har lav dropout og er kapabel til å gi strøm opp til 1.5 Ampere.

4.4.3 Kobling mot enkoder

Mellom digitalkortet og enkoderen blir signalet sendt som differensielle signaler. For å unngå jordsløyfer blir alle inngangene koblet mot optokoblere.

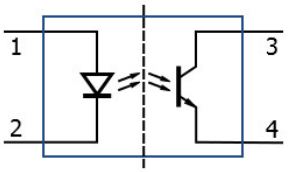
Differansedriver



Figur 4.14: Linjedriver for differensiell overføring

Differensiell signalering er en metode hvor signalene det sendes to komplimentere signaler over to linjer. Denne overføringsmetoden er som regel bedre enn å sende over en enkelt leder siden eksterne forstyrrelser bidra likt på begge linjene. En annen fordel er at det ikke er nødvendig med et referansesignal siden signalene sammenlignes mot hverandre. Linjedriveren som er brukt er en SN65LBC179AD fra Texas instruments. Den er spesifisert for hastigheter opp til 3 Mbps.

Optokobler



Figur 4.15: Overføring av elektriske signaler med optokobler

De differensielle signalene mottas av en optokobler som overfører de elektriske signalene ved hjelp av en lysdiode. Ved å bruke optokoblere får man et galvanisk skille mellom instrumentet og enkoderen. Dette gjør at instrumentet kan takle store spenningsforskjeller mellom jordplanene til instrumentet og enkoderen. Optokobleren som er brukt er en HCPL-0603.

4.4.4 FPGA

FPGAer (Field Programmable gate arrays) er digitale integrerte kretser som består av programmerbare blokker med logikk og programmerbare koblinger mellom disse blokkene. En FPGA er ikke låst til en spesifikk oppgave men kan programmeres til å utføre mange forskjellige.

Til å holde styr på signaler til og fra ADCene og enkoderen brukes det en Altera Cyclone IV FPGA. Pakkens fulle navn er EP4CE15F17C7 og den har 256 pinner hvorav 166 er IO-pinner. Pakken innholder 15000 logiske elementer og har innebygget minne på 504 Kbit. FPGAen krever minimum to spenninger, en kjernespenning på 1.2V og en IO-spenning som kan være 2.5 eller 3 volt. IO-spenningen er satt til 2.5 volt i dette designet. Kretsen trenger også en ekstern oscillator som referanseklokke. Ved oppstart laster kretsen inn en konfigurasjonsfil fra et eksternt flashminne.

4.5 Digital logikk

Den digitale logikken er kodet i VHDL for å implementeres på FPGAen. Kretsen er programmert til å styre kontrollsignalene til AD-konverterene på analogkortet, motta data fra AD-konverterene midlertidig lagre dem, og å overføre dem til rakettenkoderen.

4.5.1 Klokkesignaler

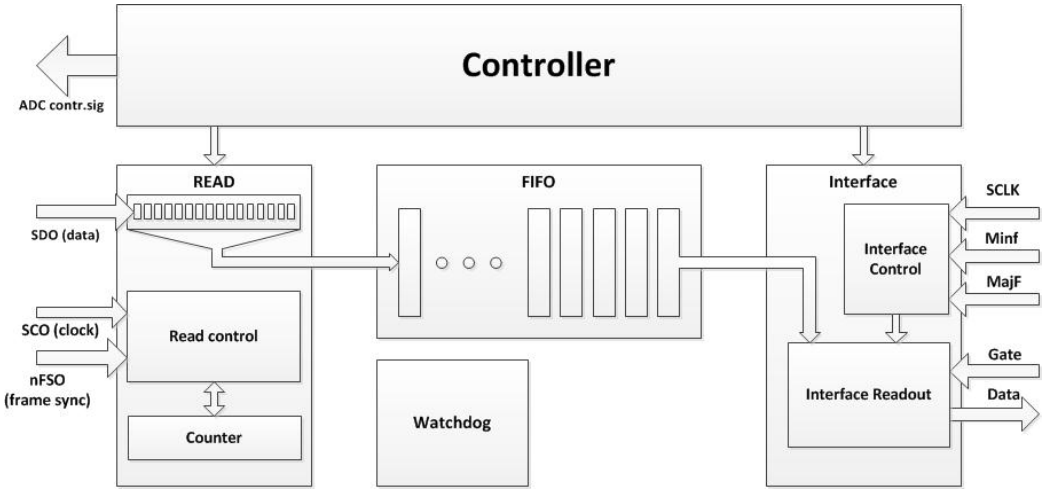
FPGAen drives av en 50MHz klokke som kommer fra en krystallosilator. Antialiasingfilteret og AD-konverterene på analogkortet trenger egne frekvenser. Disse blir generert av en PLL internt i FPGAen ut i fra systemklokka på 50 MHz.

4.5.2 Dataflyt

Den digitale logikken kan deles inn i forskjellige moduler. Hver av modulene har forskjellige oppgaver. Figur 4.16 viser en skjematisk oversikt over dataflyten i FPGAen.

Readmodulen

Den fallende flanken til $\overline{FSO}$ signaliserer starten på en ny dataoverføring fra ad-konverterene. Når de første 32 bitene er klokket inn til readmodulen vil modulen se på kontrollbitene for å se om disse er dataene er gyldige. Hvis dataene ikke er gyldige vil modulen forkaste dataene å vente til neste



Figur 4.16: Blokkskjema av dataflyt i FPGA. Data fra AD-konverterene blir skiftet inn i en lesemodul hvor de 16 mest signifikante bitene blir sendt serielt inn til midlertidig lagring i FIFO. En interfacemodul tar seg av utlesning fra FIFO og klokker verdiene ut på datalinjen til enkoderen. Kontrollmodulen styrer kontrollsignaler til ad-konverterene samt styrer lesing inn og ut til FIFO. Watchdogmodulen har kontroll på resetsignaler. En del av kontrollsignaler er utelatt for å gjøre figuren enklere.

fallende flanke. Hvis dataene er gyldige vil modulen sende de 16 øverste bitene til FIFOen. Modulen må på forhånd vite hvor mange ad-konvertere som er koblet i kjede, siden de påfølgende databitene ikke er innrammet av $\overline{FSO}$. Derfor vil hele settet med verdier bli kastet hvis den første konverteren ikke har gyldige data. Fordelen med denne metoden er at det alltid vil klokkes inn et sett med konverteringer fra alle konvertere. Det vil heller ikke bli sendt data til FIFOen hvis antallet lagrede verdier passerer en satt grense. Hvis grensen passeres midt i et sett med konverteringer vil det resterende settet bli sendt inn før modulen stopper å sende.

FIFO

FIFOen er generert av Altera sin modulgenerator å benytter seg av minnesegmentene innlagt i FPGAen. FIFOen har en W_req inngang som går høy hver gang readmodulen har satt en ny verdi på linja. FIFOen har en tilsvarende utgang som styres av interfacemodulen. Dataverdien først i køen bli automatisk satt på utgangen. Når interfacemodulen har lest verdien går R_rec høy og neste verdi blir satt ut. FIFOen har tre statussignaler: *Full*, *Empty* og *Wcount*. *Full*, *Empty* signaliserer når FIFOen er henholdsvis full eller tom.

Wcount viser hvor mange ord det til enhver tid er lagret på FIFOen.

Interfacemodulen

Interfacemodulen holder styr på utlesningen til enkoderen. Modulens oppgave er å vite hvor i telemetrirammen man befinner seg og å lese ut de riktige verdiene fra FIFOen. Alle tellere i modulen resettes hver gang *MinF* symboliserer starten på en ny ramme. Modulen legger dataene klare på et skiftregister. Når *Gate* går høyt blir disse skiftet ut bit for bit på den stigende flanken til *SCLK*.

Kontrolmodulen

Kontrolmodulen har som oppgave å holde styr på *Sync* og *nReset* signalene til AD-konverterene. Modulen kontrollerer også data og rammesignal for lesing av data inn til ad-konverteren.

Watchdogmodulen

Watchdogmodulen har som oppgave å holde styr på resetsignalene i kretsen. Disse signalene er koblet opp mot *MajF* signalene fra enkoderen slik at hele systemet resettes for hver 256. MajF-puls.

Clk_divmodulen

Denne modulen er generert av modulgeneratoren til Altera. Denne modulens oppgave er å dele ned systemklokka fra oscillatoren, til 400 KHz til antialiasingfilteret og 9 MHz til AD-konverterene på analogkortet.

4.6 Kretskortutlegg

Denne seksjonen tar for seg utlegget av elektronikken og aspekter rund dette.

4.7 Bruk av jord og spenningsplan

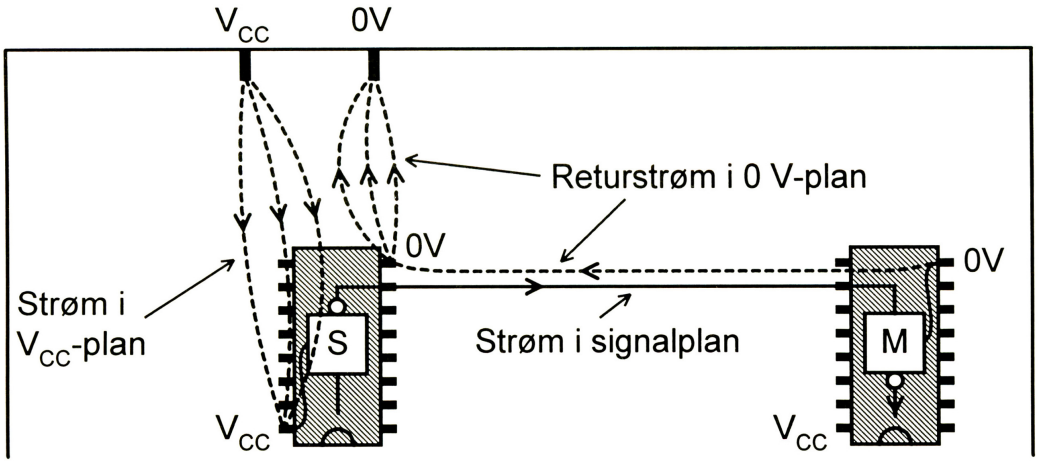
Jord og spenningsplan brukes i så å si alle mulitlagkretskort i dag. Det å bruke jordplan gir fordeler som:

- God HF-jordreferanse
- Reduserer generering av 'common mode'-stråling

- Reduserer parasittkapasitanser mellom ledere.
- Virker som skjerm

For beste effekt bør det være minst et eller flere jordplan pr. spenningsplan. Hvis konstruksjonen trenger flere spenninger kan man dele opp et spenningsplan, eller det kan benyttes et plan per spenning.

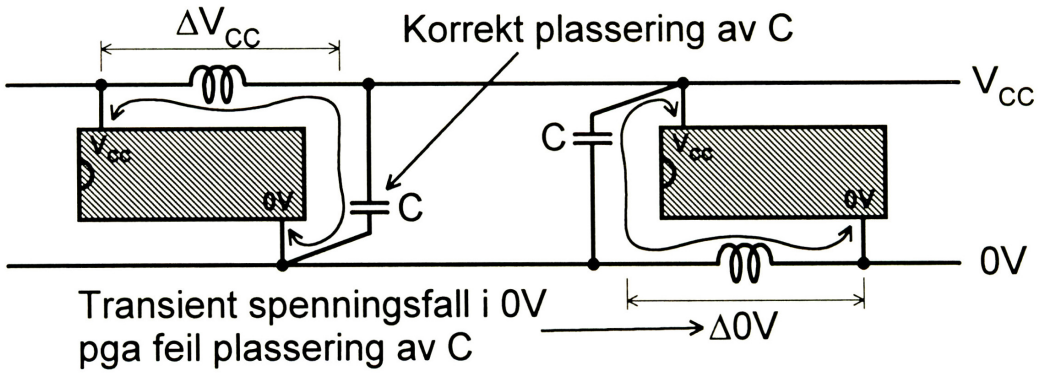
Konstrueres jord og spenningsplan riktig kan det gi en støydemping på 10 - 20 dB(Grøndal 1997). Noe av det som gjør konstruksjon av jordplan vanskelig, er at man har mindre kontroll på hvor og hvordan strømmer i planet vil bevege seg. I en vanlig leder er strømmen begrenset til denne, mens en strøm mellom to punkter i et plan vil, hvis uforstyrret, fordele seg i en vifteform siden elektronene frastøter hverandre. Returstrømmen til en signalleder vil i et jordplan velge den veien med lavest induktans. Det betyr at returstrømmen vil velge en bane som går direkte under signallederen. I store konstruksjoner er det ofte så mange som 1000 viahull som alle går gjennom jordplanet. Hvis det blir en opphoping av hull i et område kan det føre til at returstrømmer blir tvunget ut fra sin foretrukne bane. Hvis man ikke har dette i bakhodet når man konstruerer kortet, kan man risikere at returstrømmene går inn i støyfølsomme områder. For å unngå problemer kan man bruke spalter i jordplanet slik at returstrømmer blir ledet utenfor slike følsomme områder, men denne metoden er omdiskutert og bør brukes med forsiktighet siden det i utgangspunktet er vanskelig å gjette hvor returstrømmene vil gå (Grøndal 1997). Generelt er det bedre å legge ned mye arbeid i gunstig plassering av komponenter slik at man unngår bruk av spalter. Figur 4.17 viser elektronenes bevegelser i et jord og spenningsplan.



Figur 4.17: Returstrømmer i et jordplan. (Grøndal 1997)

4.8 Avkobling

For at en krets skal fungere slik den skal er det viktig at det er en stabil og god spenningstilførsel. En digital konstruksjon vil konstant ha en pulsaktivitet. Disse pulsene skaper transienter mellom konstruksjonens V_{cc}-kontakt og jord. For å beskytte integrerte kretser mot disse transientene setter man opp avkoblingskondensatorer mellom kretsens jordpinne og forsyningsspenning. En avkoblingskondensator bør plasseres så nær ICens jordpinne som mulig (Grøndal 1997). Dette fordi det blir minst mulig strømbaner i kortets jordplan. Hver IC på kortet bør ha en egen høyfrekvens avkoblingskondensator. Det bør også være minst en lavfrekvenskondensator i nærheten av kortets V_{cc}-kontakt. Verdiene på en avkoblingskondensator varierer fra IC til IC. En mye brukt verdi er på 100 nF. En lavfrekvent avkoblingskondensator ligger gjerne på noen μF, men kan i noen tilfeller komme opp i flere hundre μF.



Figur 4.18: Korrekt plassering av avkoblingskondensatorer. (Grøndal 1997)

4.9 Støybegrensing

Enhver kretskonstruksjon støyer. Det er derfor viktig å tenke støybegrensning i alle ledd av utviklingen. Når man konstruerer for støybegrensning er det i korte trekk å begrense sløyfearealer, strøm og frekvens. Sløyfearealer blir i stor grad minimert når man konstruerer multilagskort slik at man kan benytte seg av hele jord og spenningsplan. Digitale enheter er en stor kilde til støy. Steile flanker fra klokkesignaler gir mange uønskede overharmoniske frekvenskomponenter. Man bør derfor aldri bruke høyere klokkefrekvenser enn nødvendig. Som tidligere nevnt kan også returstrømmer i jordplanet lage krøll hvis de kommer inn følsomme områder. Det kan ofte være lurt å skille analog og digital jord, men det er en skjønnsmessig vurdering fra konstruksjon til konstruksjon.

4.10 Bilder av utlegg

4.10.1 Probekort

Figur 4.19 viser utlegget av probekortet. Probekortet er sirkulært med en diameter på 40 mm. kortet festes til bommen, inne i probekulen, gjennom et hull i midten av kortet. Probekortet har tre elektriske lag. Signalbanene går i topplaget. Jordlaget ligger mellom signal- og powerlaget. Probekulen er elektrisk koblet til kontakten øverst til venstre på kortet. Herfra går signalet gjennom et 2. ordens RC-filter med knekkfrekvens på 50 KHz. Så sendes signalet gjennom en opamp koblet som spenningsfølger før det sendes inn i en opamp koblet som en inverterende forsterker. Utgangen fra den andre opampen sendes til instrumentkortene gjennom en koaksialkabel koblet til kontakten øverst til høyre. Powerlaget inne i kortet er delt i to seksjoner, ett for positiv 5 volt og ett for negativ 5 volt.

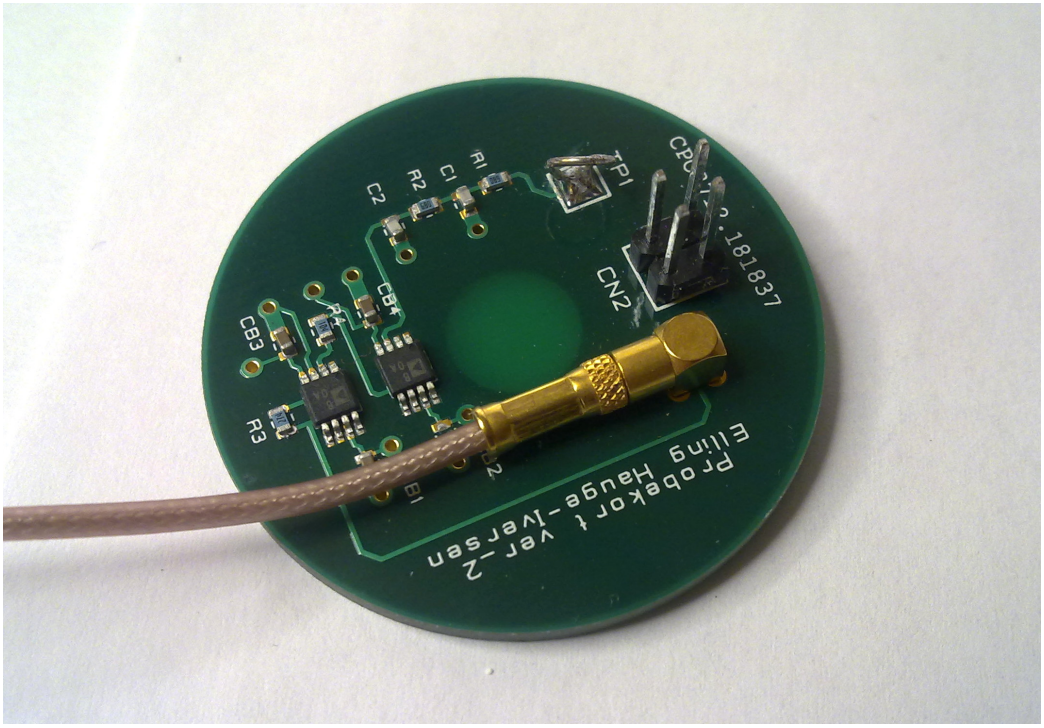
4.10.2 Analogkort

Figur 4.20 viser oversiden av analogkortet. På venstre side er det seks kontakter for tilkobling til probene. På høyre side ser man AD-konverterene og kort-til-kort-kontakten. På undersiden av analogkortet går klokkelinjen til antialiasingfilteret signallinjer med klokke, data og kontrollsignaler fra digitalkortet. Kortet har et jordplan og et spenningsplan inne i kortet. Spenningsplanet er delt mellom positiv 5 volt og negativ 5 volt. Utleggsfilene er vedlagt i tillegg E

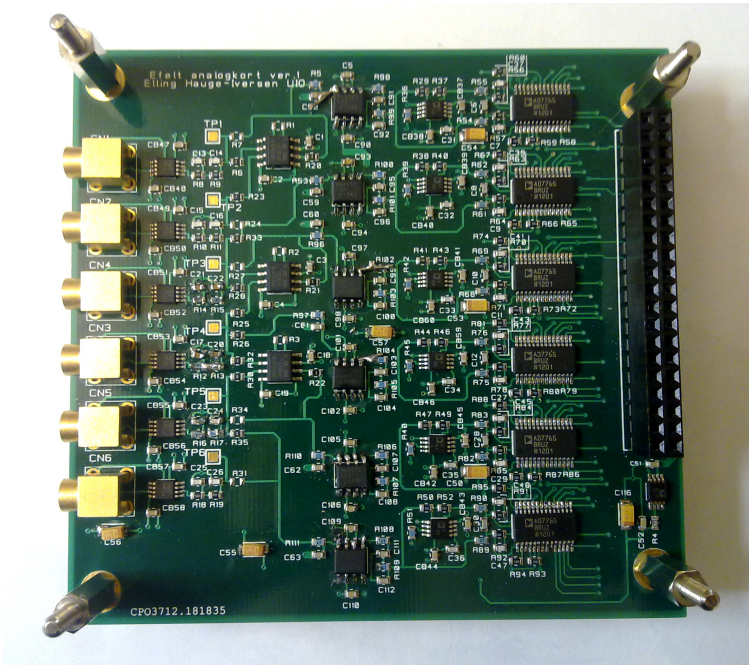
4.10.3 Digitalkort

Figur 4.21 viser over- og undersiden av digitalkortet. På høyre side på oversiden ser man kort-til-kort-kontakten. Øverst på midten er FPGAen plassert, denne får programmeringsbitfilen fra minnebrikken som er plassert øverst til høyre. Kontakten mot enkoderen sitter nederst til venstre. Gjennom denne går alle signaler samt driftsspenning på 28 volt. Nederste del av kortet består av tre spenningsomformere. Den lengst til venstre omformer fra 28V til $\pm$5V, den i midten genererer 2.5V og den til høyre 1.2V. Kontakten øverst til venstre leverer spenning og jord til de seks probene. Øverst til høyre er programmeringskontakten plassert. Undersiden til digitalkortet er delt inn i flere seksjoner med hvert sitt spenningsnivå. Seksjonen ved inngangskontakten øverst til venstre er designert enkoder-jord. Under FPGAen ligger det et lag som forsyner enheten med 1.2V. Nederst til høyre er det et lite plan som forsyner 2.5V til analogkortet gjennom kort-til-kort-kontakten. Under

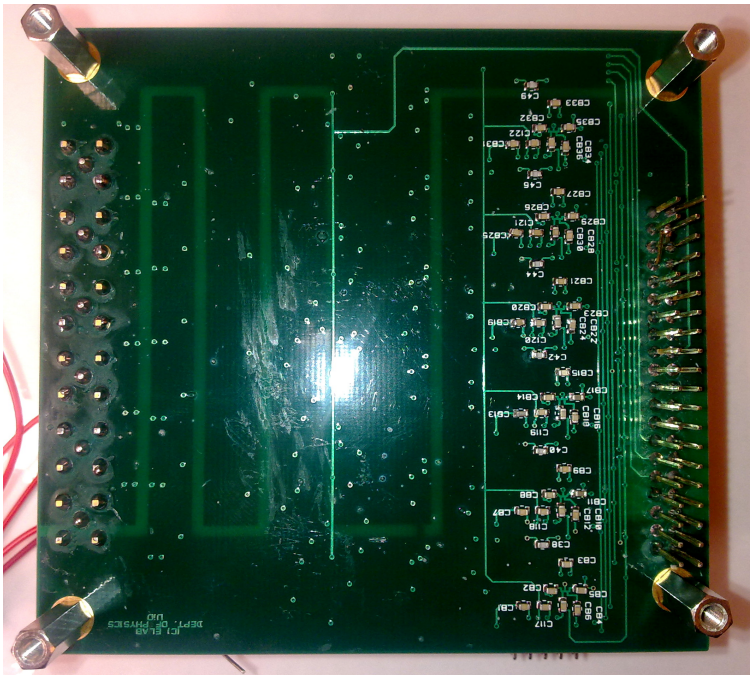
FPGA-en ligger det en rekke med 100nF avkoblingskondensatorer. Det resterende planet ligger på -5V. Utleggsfilene er vedlagt i tillegg E



Figur 4.19: Probekort

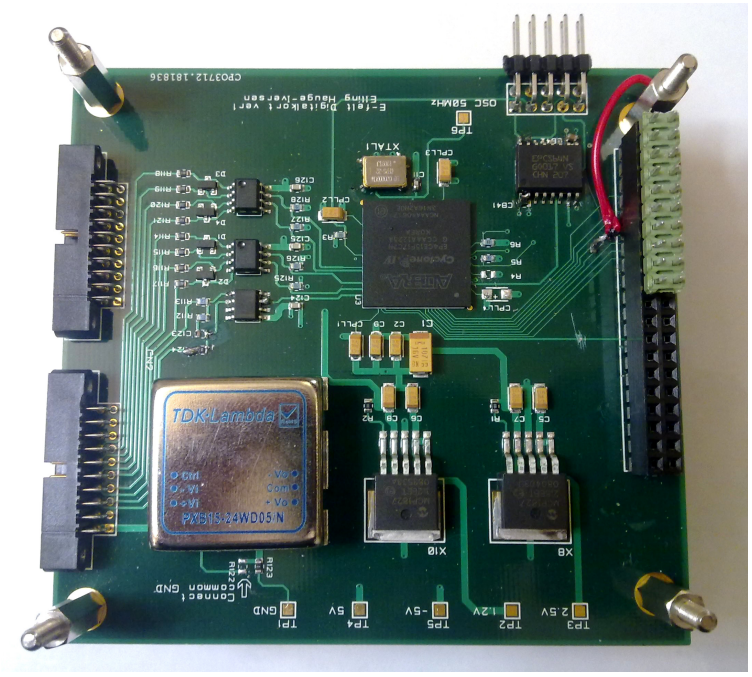


(a) Analogkort overside

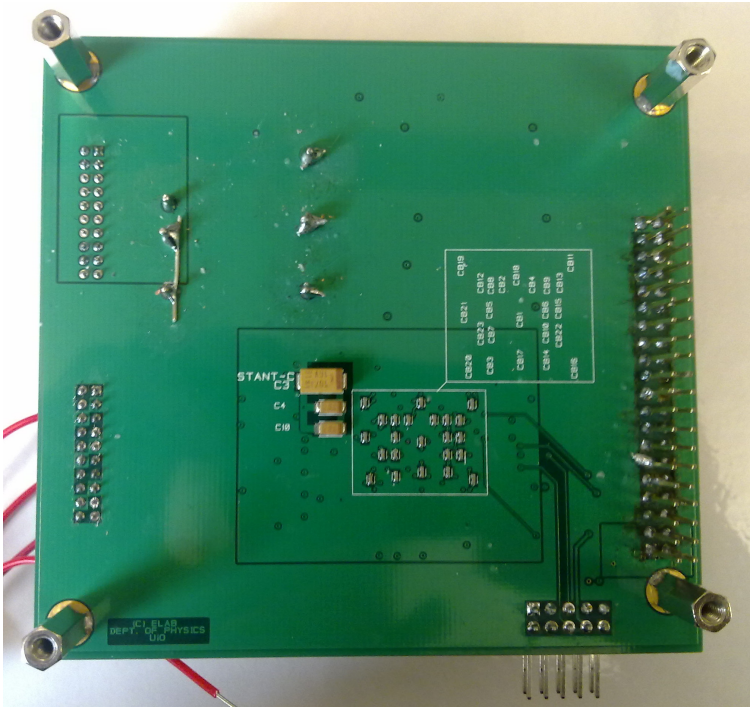


(b) Analogkort underside

Figur 4.20: Analogkort

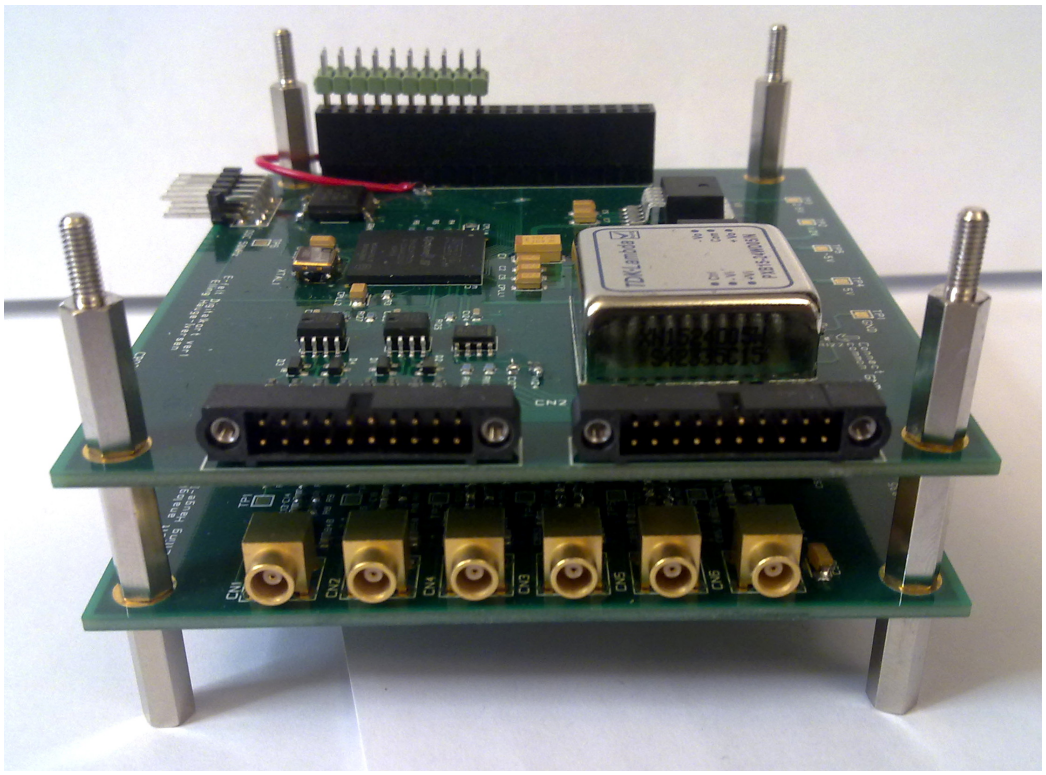


(a) Digitalkort overside

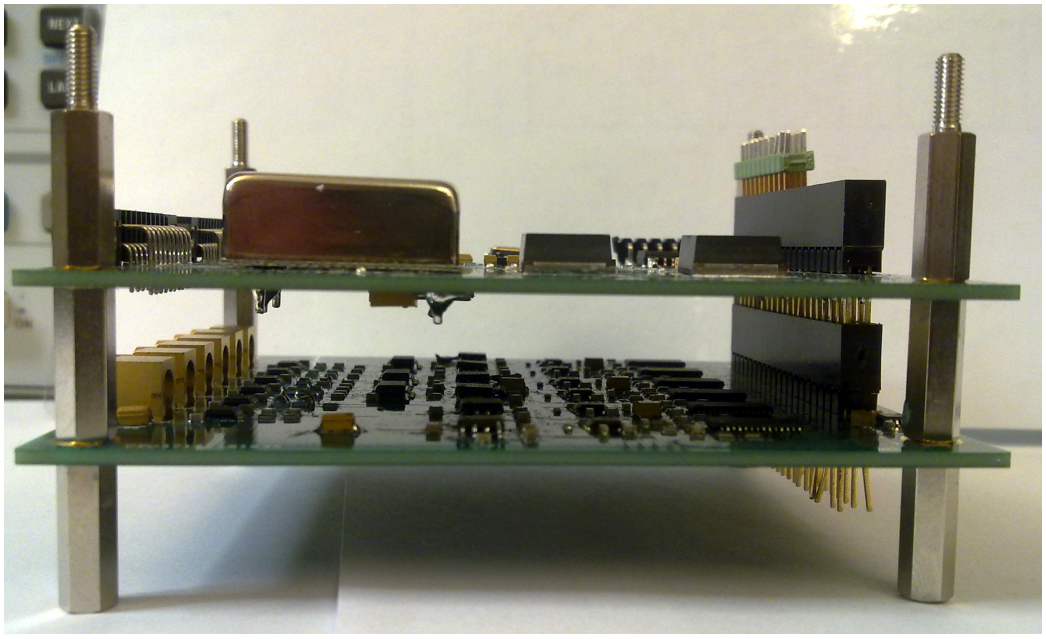


(b) Digitalkort underside

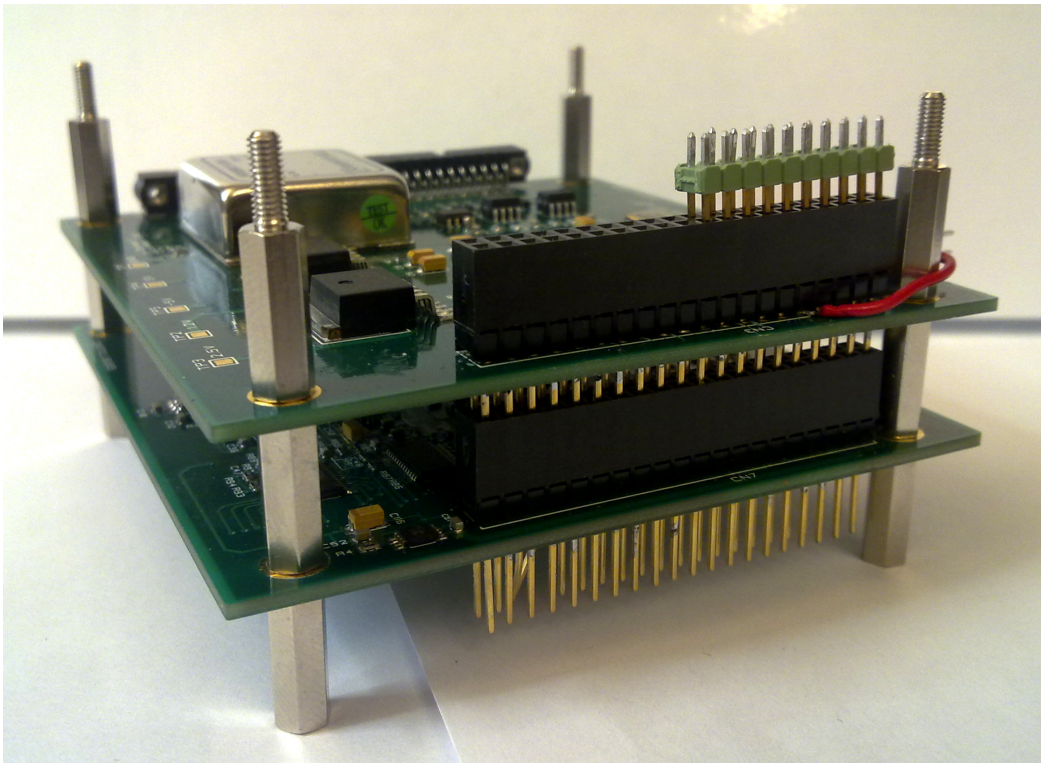
Figur 4.21: Digitalkort



Figur 4.22: Instrument forfra



Figur 4.23: Instrument side



Figur 4.24: Instrument bak

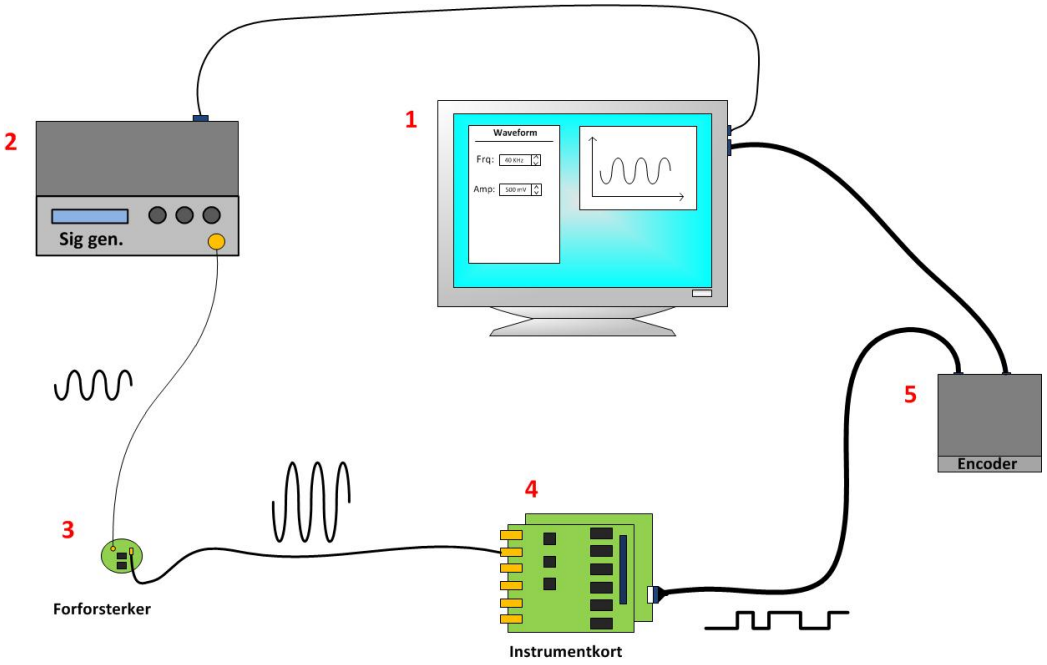
Kapittel 5

Testing og kalibrering

Testingen av e-feltinstrumentet ble gjort på elektronikklaboratoriet til rom- og plasmagruppa. Målet med testingen var å kunne fastsette at instrumentet fungerte som det skulle og ga en korrekt gjengivelse av påført stimuli innenfor frekvensområdet av interesse. Testingen skulle også gi grunnlag for en revisjon av instrumentet. Som tidligere nevnt i kapittel 2 ligger styrken til elektriske felter rundt noen titalls mV/m. Siden raketten roterer under flyvningen vil en probe som er på en 1,5 meter lang bom oppleve en potensialendring på 30 mV på en rotasjon hvis styrken på feltet er 10 mV/m. Påført stimuli gikk fra 1 Hz til 10 kHz med amplitude på 10, 50 og 100 mV.

5.1 Testoppsett

Figur 5.1 viser testoppsettet. Probekortet ble plassert ca. 1,5 meter unna instrumentkortet. Spenning på ± 5 volt og jord ble forsynt fra digitalkortet gjennom tre flettede ledninger. Signalet fra probekortet til analogkortet ble overført med en koaksialkabel med ytterkappen koblet til utgangen til spenningsfølgeren på analogkortet. Stimuli ble levert av en Agilent ag332200 signalgenerator som ble styrt gjennom et labview program. Signalgeneratorens jord ble koblet til jordpinnen på probekortet. Digitalkortet var koblet til enkoderen med en kabel som leverte en driftsspenning på 28 volt og kommunikasjonssignalene *SCLK*,*Gate*, *MinF*, *MajF* og *Data*. Enkoderen måtte programmeres med telemetrioppsettet for å kunne gi korrekte signaler på *Gate* og *SCLK*. *SCLK* gir en frekvens på 3.3 MHz og *Gate* ga pulser slik at hver kanal ble overført med en hastighet på 17 KHz. Datastrømmen fra enkoderen blir overført til PCen hvor den blir lest av og/eller lagret til fil. Antialiasingfilteret ble under testingen satt til 5 KHz. Kanalene ble testet en og en, de kanalene som var koblet til differanseforsterkerne ble først testet



Figur 5.1: Testoppsett

med probekortet koblet til den ene inngangen og jord koblet til den andre og omvendt.

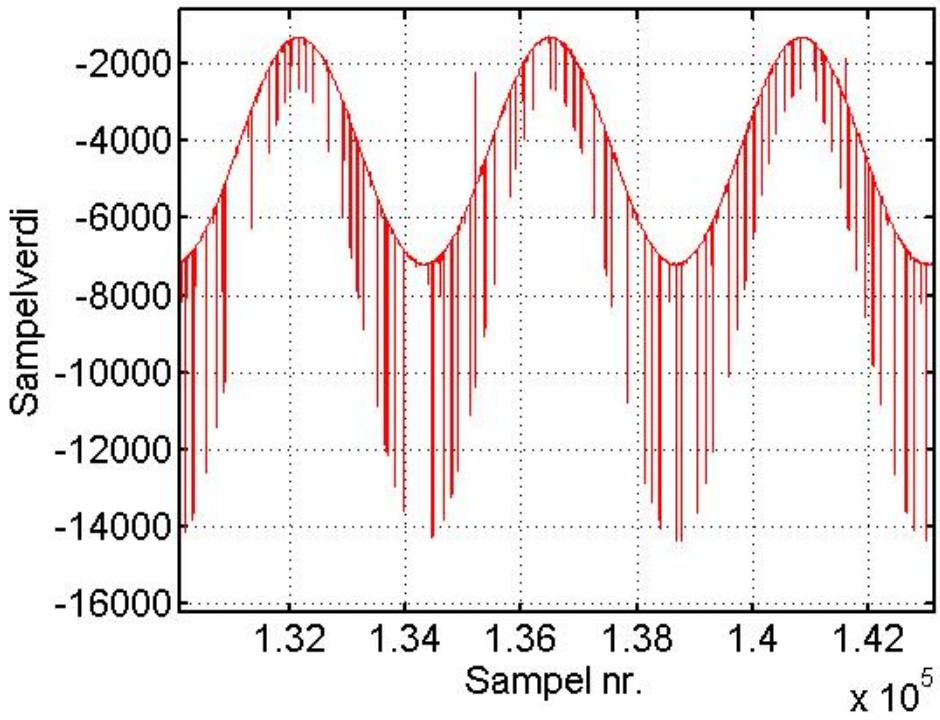
5.2 Utlesing og behandling av data

Signalgeneratoren ble styrt gjennom labviewprogrammet *Calibration_efelt.vi*. Programmet bruker en for-løkke til å steppe seg dekadisk gjennom frekvenser fra 1 Hz til 10 KHz. For å skille ut hver kanal fra enkoderens rådata ble matlabscriptet *read_eidel_tm_general_ICI4_efelt_v1.m* brukt. Det er skrevet av Espen Trondsen ved rom og plasmagruppa ved UiO. For å konvertere dataene fra toerkompliment og plotte dataene brukes matlabscriptet *efelt_time_drift_scale.m*.

5.3 Testresultater

Dropouts

Figur 5.2 er et utdrag av sampledataene og viser den påførte sinusen med en del overlagrede dropout. Denne feilen slår ut på enkelte sample og må derfor være et resultat av feil i det digitale domenet. Når FPGAen ble satt til å sende



Figur 5.2: Dropout på datalinjen

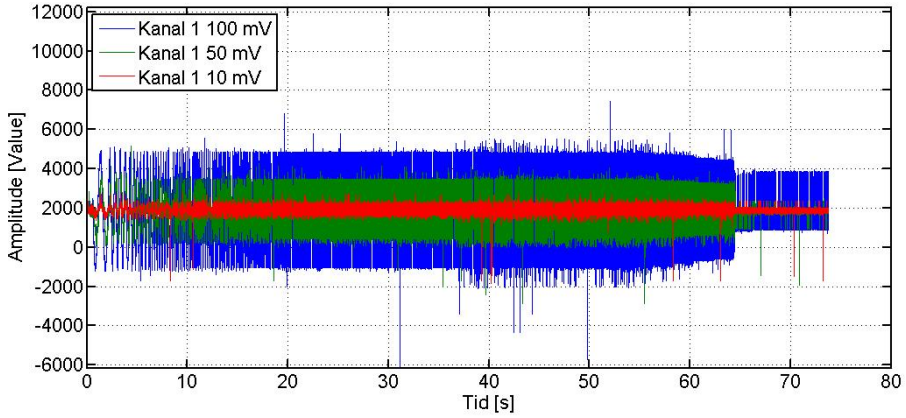
ut en konstant bitverdi viste det seg at linjedriveren i enkelte tilfeller fikk med seg overgangen fra en logisk lav til logisk høy. Figur 5.3 viser en tabell med bitverdien som skulle sendes og bitverdiene til dropoutene. Det ble først antatt at spenningsnivået til FPGAens logiske 1 lå for tett på linjedriverens grense for logisk 1 slik at skiftet kom for sent for enkoderen. I et forsøk på å komme rundt denne feilen derfor montert en nivåskifter mellom FPGAens utgang og linjedriverens inngang, for å skifte logisk høy opp til 3,3 volt, uten at dette endret antallet dropout. Det ble bestemt å bytte ut linjedriveren med en MAX490 linjedriver som er en tidligere versjon av SN65LBC179AD som var den som ble brukt. Dette førte til at dropoutene forsvant og instrumentet oppførte seg som ønsket. Det er dog ikke funnet årsaken til at den eldre MAX490 fungerte og ikke den nyere SN65LBC179AD. Dette er noe som bør være en del av en revisjon av instrumentet.

Resultater

All testing og kalibrering ble gjort før linjedriveren ble byttet til den eldre MAX490. Dette skal likevel ikke ha noe å si for den analoge delen av systemet.

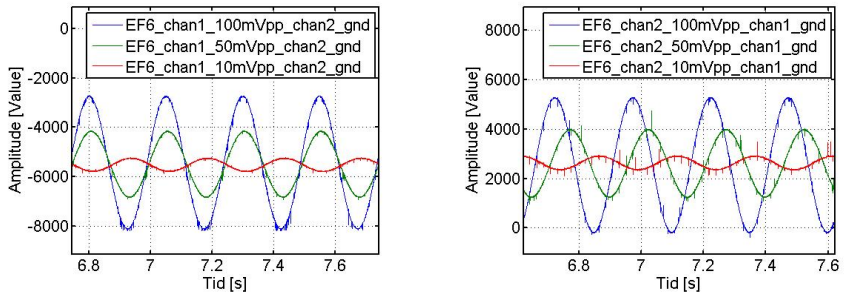
Utgang	1	0	0	0	0	1	0	1	1	0	1	1	0	0	1	0
Dropout-verdier	1	0	0	0	0	**0**	0	1	1	0	1	1	0	0	1	0
	1	0	0	0	0	1	0	**0**	1	0	1	1	0	0	1	0
	1	0	0	0	0	**0**	0	**0**	1	0	1	1	0	0	1	0

Figur 5.3: Dropoutverdier når et konstant bitmønster leses ut fra FPGAen. Øverste linjen viser den korrekte verdien, mens de tre andre viser verdiene på en hendelse. Gale bit er merket med fet type. Legg merke til at feilen skjer når datalinjen skifter fra 0 til 1.

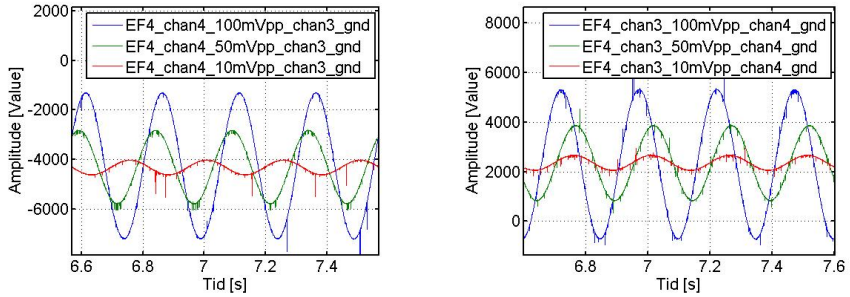


Figur 5.4: Plott av testsignalet for single ended testing av probe 5. Plottet er bearbeidet for å fjerne dropout. Tre kjøringer med amplitude på 10, 50 og 100 mV er plottet oppå hverandre.

For å få bedre figurer er datafilene er en del av dropouten fjernet ved hjelp av matlab. Figur 5.4 viser responsen for testing av single-ended måling på kanal 1. Plottet viser tre kjøringer med amplitude 10, 50 og 100 mv. Ser at antialiasingfilteret slår til ved ca. 65 sekunder i overgangen mellom 5 og 6 Khz. Amplituderesponsen er helt tydelig flat i store deler av spekteret, men faller litt av ved ca. 2KHz, noe som skyldes at antallet samples pr periode blir så lavt att det rekonstruerte signalet får toppene klippet av. Figurene 5.5 til 5.8 viser et utsnitt av testkjøringene for alle kanalene. Faseforskjellen mellom signalene skyldes at kjøringene er startet manuelt.



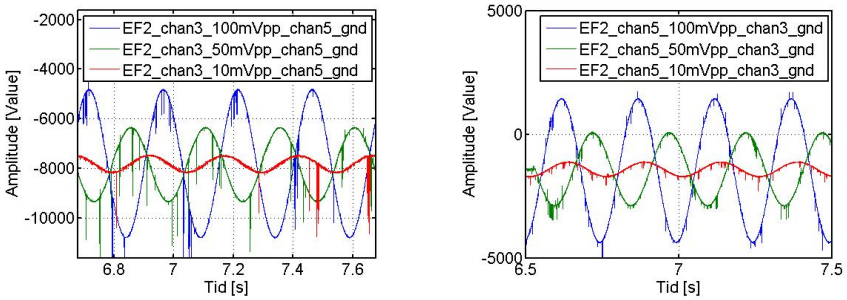
Figur 5.5: Utsnitt av testsignalet ved differansemåling mellom probe 1 og 2. På figuren til venstre er probe 2 satt til jord mens probe 1 er påsatt signal. Omvendt på figuren til høyre.



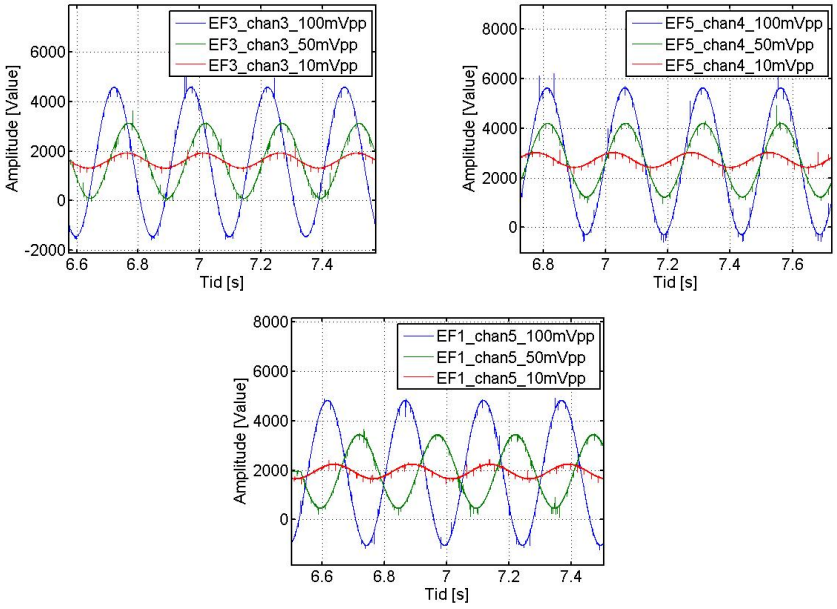
Figur 5.6: Utsnitt av testsignalet ved differansemåling mellom probe 3 og 4. På figuren til venstre er probe 3 satt til jord mens probe 4 er påsatt signal. Omvendt på figuren til høyre.

Avvik fra nullpunkt

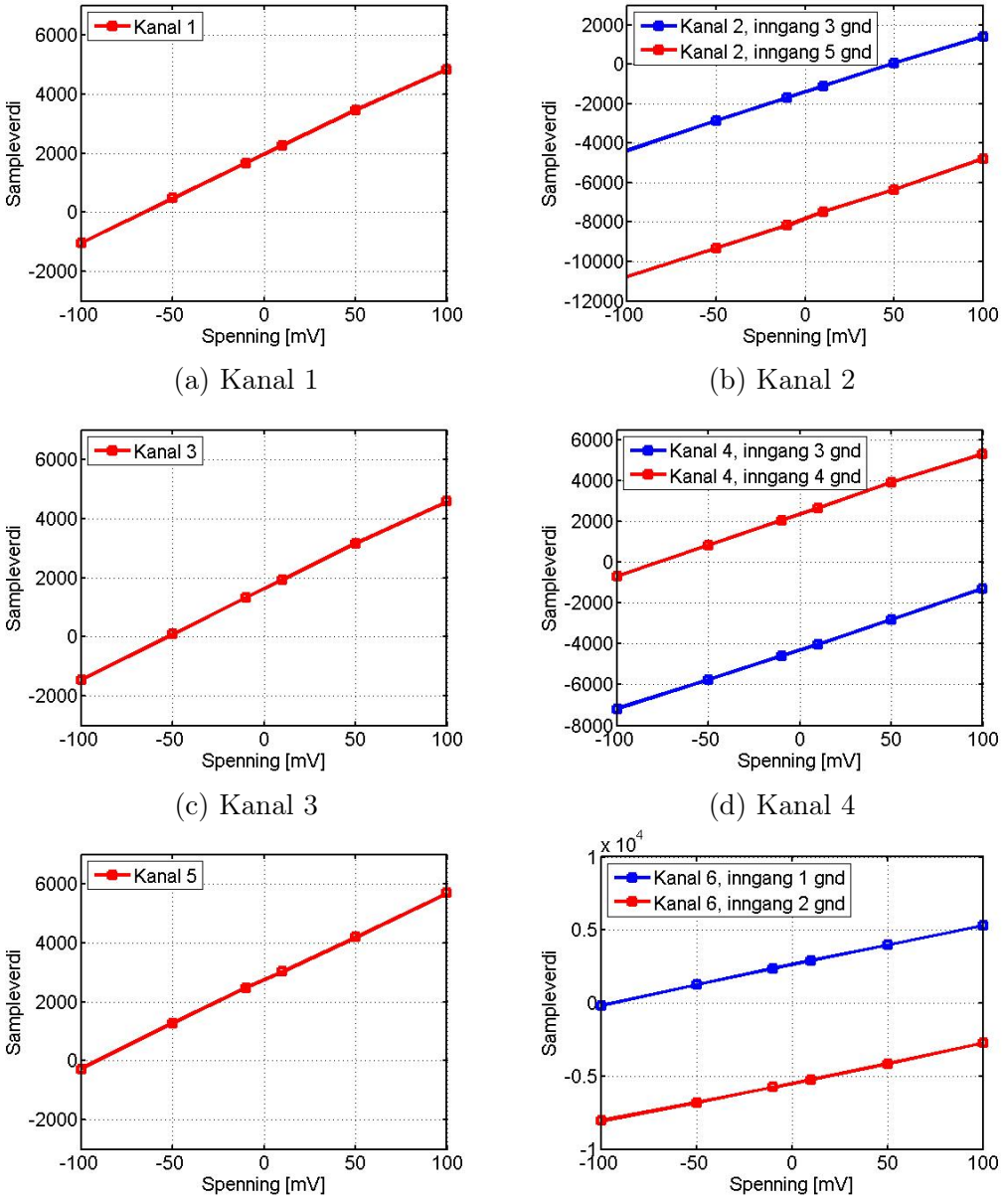
Figur 5.9 viser amplitudeutslaget for de samplede verdiene mot amplituden til påført stimuli. Figur 5.10 viser beregnet nullpunkt og avviket fra teoretisk nullpunkt. Beregningene er gjort ved å se på topp og bunnverdien til kurvene i figur 5.5 til 5.8. For de kanalene som er differensielle er nullpunktet beregnet ut i fra midtpunktene til hver av målingene.



Figur 5.7: Utsnitt av testsignalet ved differansemåling mellom probe 3 og 5. På figuren til venstre er probe 5 satt til jord mens probe 3 er påsatt signal. Omvendt på figuren til høyre.



Figur 5.8: Utsnitt av testsignalet ved single-ended målinger av probe 3, 4 og 5.



(a) Kanal 1

(b) Kanal 2

(c) Kanal 3

(d) Kanal 4

(e) Kanal 5

(f) Kanal 6

Figur 5.9: Plott av sampleamplituden mot amplituden på innsendt stimuli på alle kanalene. På de differensielle kanalene er stimuli på begge inngangene tatt med. Ser et det er en klar lineær sammenheng mellom inngang og utgang.

	Topp	Bunn	Avvik	Tilsv. spenning [mV]
Kanal 1	4842	-1046	1898	64.47
Kanal 2_1	-4810	-10780	-4630	-155.11
Kanal 2_2	1448	-4378		
Kanal 3	4575	-1432	1571.5	52.32
Kanal 4_1	-1289	-7183	-967.5	-32.83
Kanal 4_2	5318	-716		
Kanal 5	5689	-279	2705	90.65
Kanal 6_1	-2725	-8088	-1427.25	-53.23
Kanal 6_2	5271	-167		

Figur 5.10: Beregning av avvik fra teoretisk nullpunkt.

5.4 Kalibrering

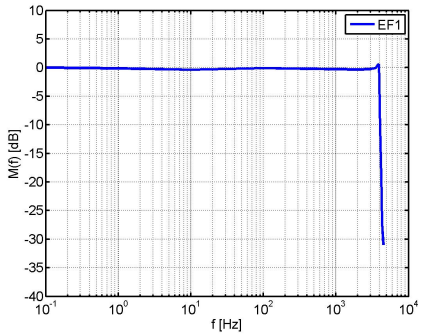
Kalibreringen har som hensikt å kartlegge hver av kanalenes individuelle karakteristikk, slik at et fremtidig måleserie kan tilpasses til en mer korrekt gjengivelse av mediet som er målt. Frekvenskarakteristikken til instrumentet forteller oss hvordan instrumentet reagerer på forskjellige frekvenser, og om noen av disse blir dempet mer enn andre. Det er også viktig å ha oversikt over hvorvidt signalets nullpunkt drifter ved forskjellig frekvens. Kalibreringen er gjort manuelt men denne prosessen er ønskelig og fullt mulig å gjøre automatisk.

5.4.1 Utlesing av data

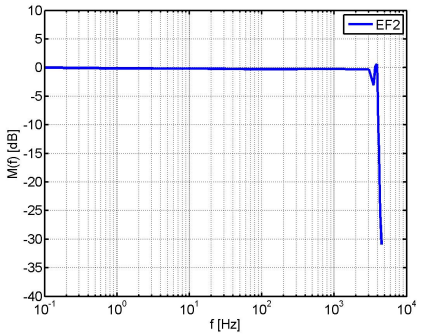
Antialiasingfilteret blir satt til å knekke ved 4 KHz. Et signal med amplitude på 100 mV ble generert av signalgeneratoren. Frekvensen ble økt stegvis for hver måling. Rundt knekkfrekvensen er målingene tatt tettere enn ellers, for å få en god gjengivelse av knekkområdet til antialiasingfilteret. Ved hver måling er topp- og bunnverdi av signalkurven notert.

5.4.2 Frekvensrespons

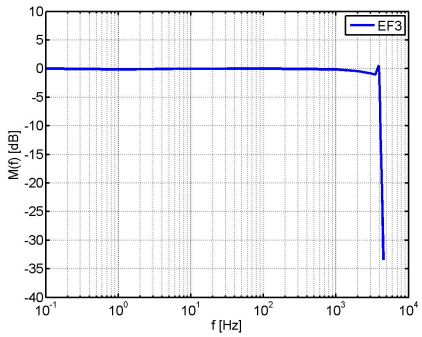
Figur 5.11 viser frekvensresponsen til alle kanalene på instrumentet. De viser en klar lavpassfilterkarakteristikk med knekk på 4 KHz, noe som er konsistent med knekkfrekvensen til antialiasingfilteret. Det er ingenting i målingene som tyder på at det er store rippler i passbåndet.



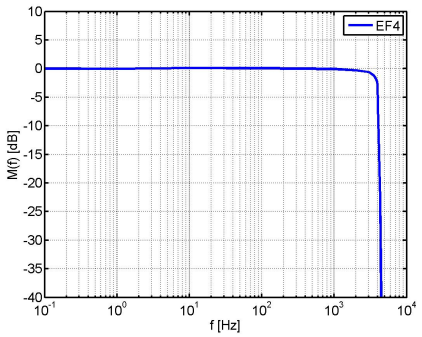
(a) Frekvensrespons kanal 1



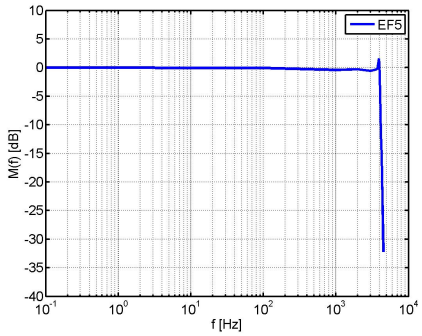
(b) Frekvensrespons kanal 2



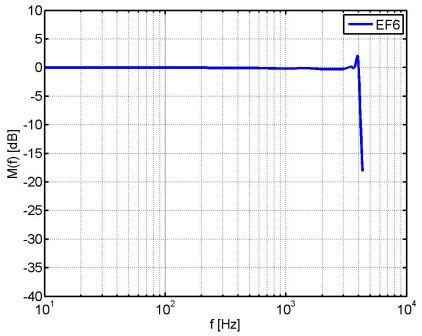
(c) Frekvensrespons kanal 3



(d) Frekvensrespons kanal 4



(e) Frekvensrespons kanal 5

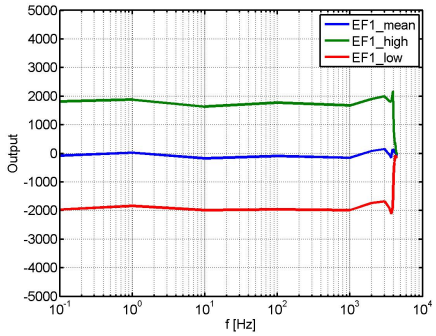


(f) Frekvensrespons kanal 6

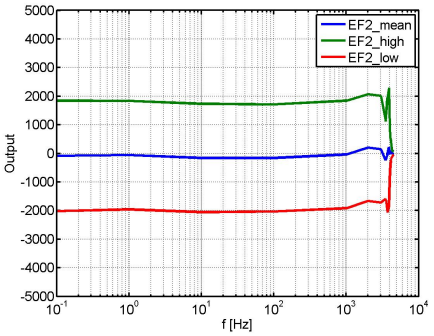
Figur 5.11: Frekvensresponsen til instrumentets seks kanaler.

5.4.3 Drift i spenningsnivåer

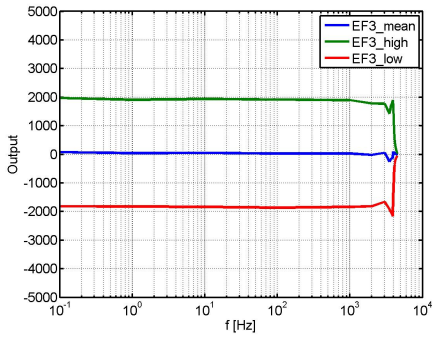
Figur 5.12 viser hvordan spenningsnivåene til kanalene holder seg jevnt over stabilt over hele frekvensspekteret, med noe unntak av rett før knekkfrekvensen på 4 KHz. Datasettet er behandlet slik at signalene i figurene svinger rundt 0.



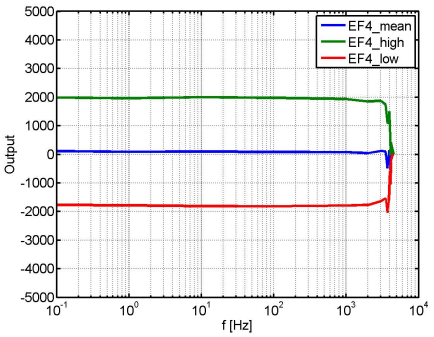
(a) Nullpunktdrift kanal 1.



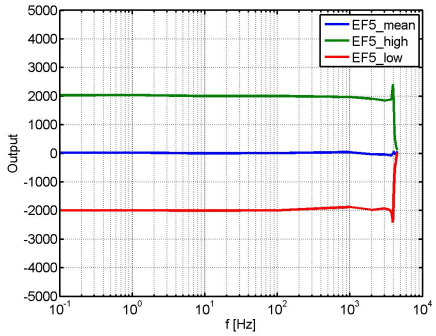
(b) Nullpunktdrift kanal 2.



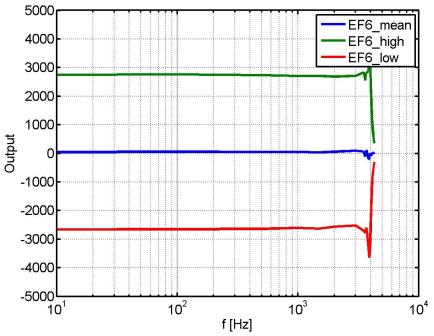
(c) Nullpunktdrift kanal 3.



(d) Nullpunktdrift kanal 4.

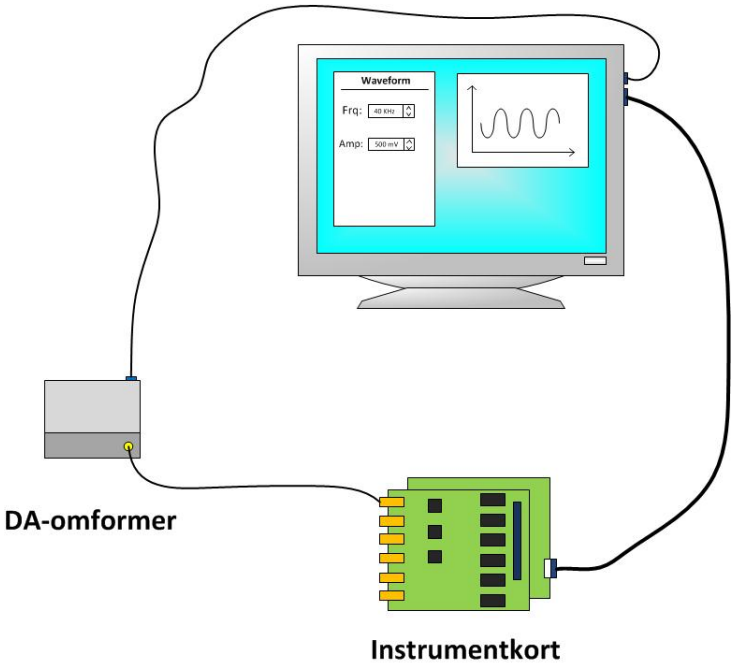


(e) Nullpunktdrift kanal 5.



(f) Nullpunktdrift kanal 6.

Figur 5.12: Drift i spenningsnivået for instrumentets seks kanaler.



Figur 5.13: Prinsipp for et PXI-basert kalibreringsoppsett. PCen styrer en DA-omformer som genererer et stimuli. Utgangen fra instrumentkortet leses av PCen.

5.4.4 Automatisering av kalibrering

Forrige seksjon tok for seg kalibrering av instrumentet. Denne prosessen tar lang tid når den blir gjort manuelt. Derfor vil det være en stor fordel å kunne kjøre en kalibreringsrutine gjennom en PC som bare kobles direkte til instrumentet. En av de største fordelene vil være at det vil være mulig å kalibrere instrumentet mens det er montert i raketten, og på den måten være sikker på at man får forhold rundt prosessen som er så identisk som en reell flyvning. Et PXI-basert kalibreringssystem programmert i Labview vil være en god løsning for dette.

Figur 5.13 viser hvordan et slikt PXI-basert kalibreringssystem kan settes opp. En PC styrer en digital-til-analogomformer. Instrumentkortet er programmert slik at datalinjen ikke lengre er styrt av klokken fra enkoderen, men isteden fra den interne klokken. Instrumentkortet generer et serielt signal etter RS422-standaren som sendes differensielt til en RS422 mottaker på PCen. Dette er en grei løsning da datalinjen allerede er koblet til en differansedriver. Alle dataene lagres til fil og kan behandles i Matlab i ettertid.

Kapittel 6

Konklusjon

Dette kapittelet summerer opp arbeidet som er gjort i denne oppgaven og ser på videre arbeid som kan og bør gjøres.

6.1 Oppsummering

Her følger noen punkter av hva som er oppnådd med oppgaven.

- Det er konstruert et e-feltinstrument til bruk på en sonderakett. Instrumentet består av fire deler, forforsterker , analogkort, digitalkort og digital logikk. Instrumentet har innganger til seks måleprober og det kan velges mellom differensielle eller single-endede målinger. Hver av probene har forforsterker plassert inne i probekulen slik at signaltap mellom plasma og elektronikk skal bli minst mulig. Kanalene blir digitalisert av seks AD-konvertere koblet sammen i daisy-chain.

- AD-konvertere og overføring av måledata til enkoder gjøres med en FPGA fra Altera.

- Instrumentet er funksjonelt testet mot en enkoder tilsvarende en som vil sitte i raketten. Testing viser at instrumentet fungerer som forventet, med lite forvrengning av signalet i passbåndet og en skarp demping av frekvenser over 4 KHz. Instrumentet opplever lite drift av spenningsnivåer ved endring av frekvens.

- Det er påvist en feil i det digital designet som gjør at den opprinnelige linjedriveren for datalinjen produserer dropouts i enkelte tilfeller når

logikken går fra lav til høy.

- Et forslag til et automatisk PXI-basert kalibreringssystem er presentert.

6.2 Videre arbeid

Som tidligere nevnt i kapittel 5 har datastrømmen ut vært utsatt for dropout. Selv om dette ble løst ved å bytte til en eldre versjon av den samme driveren er det ønskelig å utrede hvorfor dette skjer. Dette kan ha stor betydning for fremtidige instrumenter som benytter seg av denne typen kobling mellom instrument og enkoder. Det bør også vurderes om forsterkerleddet i forforsterkeren bør byttes til en ikke-inverterende forsterker.

Hvis det er et mål å ha en automatisk kalibrering av instrumentet bør det skrives en modul for seriell utlesning gjennom datalinjen for overføring med rs422-protokoll. Det bør også lages et kalibreringssystem i labview.

Referanser

ARR Payload Services (2009). *Instrument/PCM encoder interface recommendations and data handling*. ARR Payload Services.

Baker, B. (2007). Where did all the bits go? *EDN*.

Bekkeng, J. (2002). Prototypeutvikling av e-felt eksperiment for små sonderaketter. Master's thesis, University of oslo.

Bekkeng, J. K. (2007). *Prototype Development of a Low-Cost Sounding Rocket Attitude Determination System and an Electric field Instrument*. Ph. D. thesis, University of Oslo.

Carter, B. and R. Mancini (2002). Op amps for everyone third edition. *Texas Instruments*.

Ceglia, E. and W. Carey (2005). *European users guide to low gravity platforms*. Erasmus User Centre and Communication Office, ESA.

Fahleson, U. (1967). Theory of electric field measurements conducted in the magnetosphere with electric probes. *Space Science Reviews 7*, 238 – 262.

Grøndal, A. (1997). *Elektromagnetisk kompatibilitet for konstruktører*. Tapir forlag.

Gumett, D. (1998). Principles of space plasma wave instrument design. *Measurement Techniques in Space Plasmas: Fields 103*, 121.

Maynard, N. C. (2001). Electric field measurements in moderate to high density space palsmas with passive double probes. *Measurements Techniques in Space Plasmas: Fields, Amarican Geophysical union, Washington*, 13–27.

Mott-Smith, H. M. and I. Langmuir (1926, Oct). The theory of collectors in gaseous discharges. *Phys. Rev. 28*, 727–763.

Pedersen, A., F. Mozer, and G. Gustavsson (1998). Electric field measurements in a tenuous plasma with spherical double probes. *Measurements*

Techniques in Space Plasmas: Fields, Amarican Geophysical union, Washington, 1–12.

Pfaff, R., R. Holzworth, R. Goldberg, H. Freudenreich, H. Voss, C. Croskey, J. Mitchell, J. Gumbel, S. Bounds, W. Singer, and R. Latteck (2001). Rocket probe observations of electric field irregularities in the polar summer mesosphere. *GEOPHYSICAL RESEARCH LETTERS 28*(8), 1431.

Rinne, Y., J. Moen, K. Oksavik, and H. Carlson (2007). Reversed flow events in the winter cusp ionosphere observed by the european incoherent scatter (eiscat) svalbard radar. *Journal of Geophysical Research 112*(A10), A10313.

Tillegg A

Matlab-script

read_eidel_tm_general_ICI4_efelt_v1.m

```
1 %
2 % INPUT:
3 %
4 % OUTPUT:
5 %
6 % Author         : Espen Trondsen, University of Oslo
7 % Date           : 08.01.2008
8 % Last revisions:
9 % 26.01.2009
10 % 17.02.2009: ET; Added "grouping" of subcommed vector
     instrument data
11 % 26.06.2009: ET; Rearranged code for readability and
     speed, improved
12 %                 timetag for supercomm
13 % 09.07.2009: ET; Added support for calibrated values
     based on formula in
14 %                 instrument header.
15 %
16 % 16.03.2011: ET; Rewrite program to read in data in a
     matrix, one frame
17 % in each row to make it easier to merge data from
     different
18 % antennas/TM-systems to fill data gaps. Data/channel
     selection will also
19 % be easier.
```

```
20
21 % This program reads a binary telemetry file from the EIDEL decoder system.
22 % It is assumed that all additional time tagging in the decoder system is off,
23 % i.e the file contains only the data values.
24 %
25 % Highlights:
26 %-----------------
27 % Removes corrupted frames (wrong length)
28 % Posibility for sub- and supercommutated channels
29 % Possibility to mask any bitpattern from 16-bit words
30 %
   ------------------------------------------------------------------------------

31
32 % Filen basert på ICI2_v3-versjonen. Tilpasset til ECOMA og filer fått fra
33 % Tom/Ulf
34
35 % TODO: bitmaskshift/autoshift
36 % TODO: generalisere rundt group. nå er den fiksert til gruppe på tre + se
37 % på forholdet startframe/group i initialisering
38 % TODO: Les inn flere filer for å fylle datagap
39 % TODO: Legend automatisk
40 % TODO: sjekk start av timetag. skal være null
41 % TODO: Ved plotting av solsensor, plot målinger og innfyllinger i
42 % forskjellig farge. Kutt på slutten og begynnelsen. Prosentvise outliers?
43 % TODO: Mission og instrumentparametre i egen config fil.
44 % TODO: Plott som funksjon av høyde i stedet for, eller tillegg til, tid
45 % TODO: Datakvalitet. Plott drops/s for hver datakilde og evt merged. Finn
46 % bredde og avstand mellom dropp for å se etter systematsike feil.
47 % TODO: Mulighet for samtidig utlesing av multikanalsinstrumenter,
```

```
48 % inkludert kalibreringskonstanter.
49 % TODO: Fjerne neg_diff? Ved å endre PCM-format til
       sync til slutt vil
50 % denne nødvendigheten bli borte.
51
52 % Close all figures and clear workspace to prepare for
       new data proscessing
53 close all;
54 %cl;
55 disp('Old workspace and figure(s) cleared!');
56 disp('Processing new data...');
57 tic
58
59  mission = 'ICI4';
60
61
62 instr = 'EFELT';
63
64 %------------------------- Mission parameters
       -------------------------------------
65 switch mission
66     case 'ICI4'
67         % data_file = 'tx2-1205LAUA.DAT';
68         data_file = 'EF4_CHAN4_10MVPP_CHAN3_GND.DAT';
69         frames_per_format = 64;
70         words_per_frame = 72;             % Number of
               words in one frame
71         frame_freq = 2893.5;
72         syncword = hex2dec('EB90');
73
74         %% Framecounter and format counter. These are
               allways processed for
75         % timing purposes. Framecounter is also used to
                extract subcom channels.
76         framec_word = 2;
77         framec_bitmaskshift = [bin2dec('0011 1111 0000
               0000') -8];
78         formatc_word = 71;
79
80     case 'ECOMA7'
81         % data_file = 'D:\ECOMA789\TM_files\ECOPB07.dat
```

```
            '; % ARS merged
82          data_file = 'chan5_test.dat'; % 10-feet
83          frames_per_format = 64;
84          words_per_frame = 48;
85          frame_freq = 1085;
86          syncword = hex2dec('EB90');
87
88          %% Framecounter and format counter. These are
                allways processed for
89          % timing purposes. Framecounter is also used to
                extract subcom channels.
90          framec_word = 2;
91          framec_bitmaskshift = [bin2dec('0011 1111 0000
                0000') -8];
92          formatc_word = 17;
93
94      case 'ECOMA8'
95          % data_file = 'D:\ECOMA789\TM_files\ECOPB08.dat
                '; % ARS merged
96          data_file = 'ECOMA8.dat'; % 10-feet
97          frames_per_format = 64;
98          words_per_frame = 48;
99          frame_freq = 1085;
100         syncword = hex2dec('EB90');
101
102         %% Framecounter and format counter. These are
                allways processed for
103         % timing purposes. Framecounter is also used to
                extract subcom channels.
104         framec_word = 1;
105         framec_bitmaskshift = [bin2dec('0011 1111 0000
                0000') -8];
106         formatc_word = 17;
107
108     case 'ECOMA9'
109         % data_file = 'D:\ECOMA789\TM_files\ECOPB09.dat
                '; % ARS merged
110         data_file = 'ECOMA9.dat'; % 10-feet
111         frames_per_format = 64;
112         words_per_frame = 48;
113         frame_freq = 1085;
```

```
114         syncword = hex2dec('EB90');
115
116         %% Framecounter and format counter. These are
                allways processed for
117         % timing purposes. Framecounter is also used to
                extract subcom channels.
118         framec_word = 1;
119         framec_bitmaskshift = [bin2dec('0011 1111 0000
                0000') -8];
120         formatc_word = 17;
121
122     otherwise
123         disp('Unknown mission. PCM format not defined!'
                );
124 end
125 % ------------------------- End mission parameters
    ------------------------------
126
127 % ------------------------- Instrument parameters
    ------------------------------
128 switch upper(instr)
129
130     case 'EFELT'
131         comm = 6;
132         startword = 7; % Ch1: 1, CH2 : 3, CH3 : 5, CH4
                : 7, CH5 : 9, CH6 : 11
133
134
135     case 'DC'
136         comm = 1/16;
137         startword = 7;  % DC1&DC2: 7, DC3: N/A
138         startframe = 14; % DC1: 14, DC2: 15
139         cal = '(instr_data-32752).* 4.84e-3'; %
                DC1result in mV/m
140         % cal = '(instr_data-33698).* 4.84e-3'; % DC2
                result in mV/m
141
142     case 'AC'
143         comm = 9;
144         startword = 7;  % AC1:7, AC3: 11
145         cal = '(instr_data-32776).* 1.13e-3'; % result
```

```
            in mV/m
146
147     case 'IRU'
148         comm = 1/4;
149         startword = 44; % 8: rate, 44: temp
150         startframe = 1; % 1: roll, 2: pitch, 3: yaw
151         group = [1 2 3];
152         bitmaskshift = [bin2dec('1111 1111 1111 0000')
                -4];
153         cal = '0.000826.*instr_data.*2.^4 -2.2';
154         leg = [];
155
156     case 'MAG'  % Magnetometer & Accelerometer, ARR
157         comm = 1/8;
158         startword = 55;
159         startframe = 0; % 0: mag_x, 1: mag_y, 2: mag_z,
                3: mag_ref
160         group = [0 1 2];
161         %           cal_x = -2.45*instr_data + 67128;
162         %           cal_y = 3.53*instr_data - 120754;
163         %           cal_z = 5.19*instr_data - 163398;
164
165     case 'DSS1_W'
166         comm = 2;
167         startword = 12; % 12: dss1, 16: dss2
168         bitmaskshift = [bin2dec('1111 1000 0000 0000')
                -11];
169
170     case 'DSS1_C'
171         comm = 2;
172         startword = 12;
173         bitmaskshift = [bin2dec('0000 0111 1111 1111')
                0];
174
175     case 'DSS2_W'
176         comm = 2;
177         startword = 16;
178         bitmaskshift = [bin2dec('1111 1000 0000 0000')
                -11];
179
180     case 'DSS2_C'
```

```
181         comm = 2;
182         startword = 16;
183         bitmaskshift = [bin2dec('0000 0111 1111 1111')
                0];
184
185     case 'NLP_ECOMA7'                   % 4-NLP Langmuir,
            ECOMA7
186         comm = 1;
187         startword = 24; % Ch1-4: 9, 10, 13, 14
188         % cal = '1.0227.*(instr_data-31821).*
                20./(2.^16)./10e6./2'; %CH1
189         % cal = '1.0095.*(instr_data-31416).*
                20./(2.^16)./10e6./2'; %CH2
190         % cal = '1.0065.*(instr_data-32411).*
                20./(2.^16)./10e6./2'; %CH3
191         % cal = '1.0089.*(instr_data-32292).*
                20./(2.^16)./10e6./2'; %CH4
192
193     case 'NLP_ECOMA8'                   % 4-NLP Langmuir,
            ECOMA8
194         comm = 3;
195         startword = 13; % Ch1-4: 9, 10, 13, 14
196         cal = '1.0094.*(instr_data-31749).* 20./(2.^16)
                ./10e6./2'; %CH1
197         % cal = '1.0076.*(instr_data-31697).*
                20./(2.^16)./10e6./2'; %CH2
198         % cal = '1.0141.*(instr_data-32229).*
                20./(2.^16)./10e6./2'; %CH3
199         % cal = '1.0117.*(instr_data-31729).*
                20./(2.^16)./10e6./2'; %CH4
200
201     case 'NLP_ECOMA9'                   % 4-NLP Langmuir,
            ECOMA9
202         comm = 3;
203         startword = 14; % Ch1-4: 9, 10, 13, 14
204         % cal = '1.0205.*(instr_data-31473).*
                20./(2.^16)./10e6./2'; %CH1
205         % cal = '1.0137.*(instr_data-32295).*
                20./(2.^16)./10e6./2'; %CH2
206         cal = '1.0124.*(instr_data-31798).* 20./(2.^16)
                ./10e6./2'; %CH3
```

```
207         % cal = '1.0140.*(instr_data-31855).*
                20./(2.^16)./10e6./2'; %CH4
208
209     case 'FBLP_AC'
210         comm = 4;
211         startword = 3; % Low: 2, High: 3
212         bitmaskshift = [bin2dec('1111 1111 1111 0000')
                -4]; % Not confirmed
213
214     case 'FBLP_DC'
215         comm = 1;
216         startword = 28; % Low: 28, High: 36
217         bitmaskshift = [bin2dec('1111 1111 1111 0000')
                -4]; % Not confirmed
218
219     case 'EC2008_FFI_HF'
220         comm = 1/32;
221         startword = 16;
222         startframe = 1;
223         % bitmaskshift = [bin2dec('0000 0000 1111
                1111') 0];
224         % bitmaskshift = [bin2dec('1111 1111 0000
                0000') -8];
225
226     otherwise
227         disp(['Sorry, "', instr, '" is not a valid
                instrument selection. Review your settings.'
                ]);
228         break;
229 end
230 %------------------- End instrument parameters
    ------------------------------------
231
232
233 % Read TM-data file and save data as uint16 array,
      little-endian
234 fid = fopen(data_file, 'r');
235 data = fread(fid, inf, 'uint16=>uint16', 'l');
236 fclose(fid);
237
238 % Find syncwords. Keep only those who are followed by a
```

```
    syncword
% words_per_frame later. Discard the rest as they are not in a frame of
% correct length
sync_idx = find(data==syncword);
sync_idx = sync_idx(diff(sync_idx)==words_per_frame);

% --- Create unique frame identifier based on frame and format counter ---

% Pick the frame- and formatcounter from the data. Apply bitmask.
framec = double(data(sync_idx+framec_word));
framec = bitshift(bitand(framec, framec_bitmaskshift(1)), framec_bitmaskshift(2));
formatc = double(data(sync_idx+formatc_word));

% OBS, kan ha spikes innimellom likevel. Kanskje like gjerne sette til NaN?
% Fjern alle negative før positiv i formatteller, kun nødvendig der
% formatteller er i siste ord, pga feil i encoder
neg_diff=find(diff(formatc)==-1) + 1;
formatc(neg_diff)= formatc(neg_diff)+2;

% OBS brke/teste på ~=0 allerede her?
% Fjern alle gjenstående spikes større enn 1 og negative
formatc(find(diff(formatc)>1)+1) = NaN;
formatc(find(diff(formatc)<0)+1) = NaN;
% TODO fjerne slengere på slutten også?

% legg sammen formatc og framec. Vær obs på at framec er forskjøvet én
% ramme (pgs sync til slutt, ikke først. Da blir vektoren en enhet for
% kort, så legg til en NaN på slutten
timetag = 64.*formatc(1:numel(formatc)-1) + framec(2:numel(framec));
timetag = [timetag; NaN];

```

```
269 %fjerne alle uregelmessigheter. timetag SKAL øke med 1,
        ellers brudd
270 non_lin = find(diff(timetag)~=1) + 1;
271 timetag(non_lin)=NaN;
272
273 % Timetag basert på timing for hver frame. Her
        kompenseres for at det kan
274 % være flere samples per frame, supercomm. Opprett
        time_veg som en
275 % mellomregningsvariabel. Bytte den tilbake til timetag
        etterpå igjen.
276 if comm > 1
277     time_vec = NaN(comm*numel(timetag), 1);
278     for k = 1:comm
279         time_vec(k:comm:numel(time_vec)) = timetag*comm
                + (k-1);
280     end
281     timetag = time_vec;
282 end
283
284 toc
285 % --- Extract selected instrument data ---
286
287 % Create an array for the selected instrument data.
        Ceil is used to round up in case of
288 % subcomm. If subcomm, further datamasking is done
        later in if-condition
289 instr_data = NaN(numel(sync_idx).*ceil(comm), 1);
290 for k=1:ceil(comm)
291     instr_data(k:ceil(comm):numel(instr_data))= ...
292         data(sync_idx + (k-1).*words_per_frame./ceil(
              comm) + startword);
293 end
294
295 % If current instrument doesn't use all 16 bit of data
        in a word, masking
296 % is done here
297 if exist('bitmaskshift', 'var')
298     instr_data = bitshift(bitand(instr_data,
          bitmaskshift(1)), bitmaskshift(2));
299 end
```

```
300
301 % Special handling to extract subcom data.
302 if comm < 1
303     % Frame numbers holding requested data
304     frame_idx = startframe:inv(comm):(frames_per_format
        -1);
305
306     data_idx = [];
307     for frame = frame_idx
308         data_idx = [data_idx; find(framec==frame)]; %#
            ok<AGROW>
309     end % for frame
310
311     % As the index created in previous for-loop is not
        in concurrent sequence, do
312     % sorting before extracting data.
313     data_idx = sort(data_idx);
314     % ved grupperte data - altså i praksis kun ved
        subcomm vektordata
315     % Ellers kan linjene under "else" kjøres for en
        helt enkel versjon.
316     if exist('group', 'var')
317         % 2 skal egentlig være lenght(group)-1
318         data_idx = data_idx(1: numel(data_idx)-2);
319         idx = data_idx(logical(framec(data_idx+1)==
            framec(data_idx)+1 & framec(data_idx+2)==
            framec(data_idx)+2));
320         timetag = timetag(idx);
321         instr_group( : ,1) = instr_data(idx);
322         instr_group( : ,2) = instr_data(idx+1);
323         instr_group( : ,3) = instr_data(idx+2);
324
325         notnan = ~isnan(timetag);
326         timetag = timetag(notnan);
327         instr_data = instr_group(notnan, : );
328     else
329         instr_data = instr_data(data_idx);
330         timetag = timetag(data_idx);
331     end
332 end % if comm<1
333
```

```
334 %% kutter instr_data og timetag ved start og 520 sek for å få kortere variable
335 %% å jobbe med. Spesielt viktig ved plotting! OBS på denne intil videre. Verifikasjon?
336 % format1 = find(timetag>63, 1, 'first') - 64;
337 % format_end = find(timetag>520*frame_freq, 1, 'first');
338 % instr_data = instr_data(format1:format_end);
339 % timetag = timetag(format1:format_end);
340
341 %% kutter instr_data og timetag ved start og 520 sek for å få kortere variable
342 %% å jobbe med. Spesielt viktig ved plotting! OBS på denne intil videre. Verifikasjon?
343 % format1 = find(timetag>(70*frame_freq*comm), 1, 'first');
344 % format_end = find(timetag>(300*frame_freq*comm), 1, 'first');
345 % instr_data = instr_data(format1:format_end);
346 % timetag = timetag(format1:format_end);
347
348 % Data will probably be multiplied with calibration factor later, so
349 % casting to double is done here
350 instr_data = double(instr_data);
351
352 if exist('cal', 'var')
353     instr_data = eval(cal);
354 end
355
356 if exist('leg', 'var')
357     legend('Roll temp', 'Pitch temp', 'Yaw temp', 'Location', 'NorthWest')
358 end
359
360 toc
361 % Plot the data, ceil(comm) to handle subcomm
362 figure
363 plot(timetag/(frame_freq*ceil(comm)), instr_data), grid % Selected signal
364 plot(instr_data)
```

```
365
366 % ---------- Add spesific instrument related post
        processing here ----------
367 %
    ------------------------------------------------------------------------


368
369
370 switch upper(instr)
371     case 'DSS1_W'
372         % Mer avansert versjon for å detektere solpuls.
                Stiller krav til bredde og
373         % "stabilitet" til pulsen.
374         spike_start = [];
375         for k = 1:numel(instr_data)-1
376             if instr_data(k)==0 && instr_data(k+1)==0
                   && instr_data(k+2)>1 && instr_data(k+3)
                   >1
377                 spike_start = [spike_start k];
378             end
379         end
380
381         % plukk ut manuelt 90-530 s
382         time_c = timetag(spike_start+9);
383         % Some of time_c is NaN. Remove them:
384         time_c = time_c(~isnan(time_c));
385
386         figure
387         plot(time_c/comm/frame_freq, 0, 'ko'), grid on;
388
389         figure, plot(diff(time_c)), drawnow
390
391         % find(time_c/2/2893.5>90)
392         % find(time_c/2/2893.5<530)
393         % time_c = time_c(7:1367);
394
395         miss_one_idx = sort(find(diff(time_c)>3500 &
              diff(time_c)<3600), 'descend');
396         for k = 1:numel(miss_one_idx)
397             time_c = [time_c(1:miss_one_idx(k)); ...
398                 NaN; ...
```

```
399                 time_c(miss_one_idx(k)+1:numel(time_c))
                        ];
400         end
401         % Fyll inn fornuftig der det står NaN;
402         n = find(isnan(time_c));
403         time_c(n) = ceil((time_c(n-1) + time_c(n+1))/2)
                ;
404
405
406         miss_two_idx = sort(find(diff(time_c)>5250 &
                diff(time_c)<5400), 'descend');
407         for k = 1:numel(miss_two_idx)
408             time_c = [time_c(1:miss_two_idx(k)); ...
409                 [NaN; NaN]; ...
410                 time_c(miss_two_idx(k)+1:numel(time_c))
                        ];
411         end
412         % Fyll inn fornuftig der det står NaN;
413         time_c_nan_idx = find(isnan(time_c));
414         for k = 1:numel(time_c_nan_idx)
415             time_c(time_c_nan_idx(k)) = time_c(
                    time_c_nan_idx(k)-1) + 1773;
416         end
417
418
419         miss_three_idx = sort(find(diff(time_c)>7050 &
                diff(time_c)<7150), 'descend');
420         for k = 1:numel(miss_three_idx)
421             time_c = [time_c(1:miss_three_idx(k)); ...
422                 [NaN; NaN; NaN]; ...
423                 time_c(miss_three_idx(k)+1:numel(time_c
                        ))];
424         end
425         % Fyll inn fornuftig der det står NaN;
426         time_c_nan_idx = find(isnan(time_c));
427         for k = 1:numel(time_c_nan_idx)
428             time_c(time_c_nan_idx(k)) = time_c(
                    time_c_nan_idx(k)-1) + 1773;
429         end
430
431
```

```
432         tic
433         ind_arr = NaN(numel(time_c), 1);
434         for i = 1:numel(time_c)
435             ind = find(timetag == time_c(i));
436             if numel(ind) == 1
437                 ind_arr(i) = ind;
438             end
439         end
440         % Fjern alle tilfeller av NaN fordi denne
                brukes til indexering
441         ind_arr = ind_arr(~isnan(ind_arr));
442         toc
443
444         plot(time_c/comm/frame_freq, 0.5, 'ro'), grid
                on;
445
446         save ind_arr ind_arr
447         % Lagre ind_arr. Den brukes også for å plukke
                ut dss_c data.
448
449         % disp('No post processing for this instrument
                .');
450     otherwise
451         disp('Nothing here yet');
452 end
453
454 clearvars -except instr_data timetag time_c
```

efelt_time_drift_scale.m

```
clear 'all';
close 'all';

data1 = 'EF1_chan5_100mVpp';
data2 = 'EF1_chan5_50mVpp';
data3 = 'EF1_chan5_10mVpp';

% data1 = 'EF2_chan5_100mVpp_chan3_gnd';
% data2 = 'EF2_chan5_50mVpp_chan3_gnd';
% data3 = 'EF2_chan5_10mVpp_chan3_gnd';
%
% data1 = 'EF2_chan3_100mVpp_chan5_gnd';
% data2 = 'EF2_chan3_50mVpp_chan5_gnd';
% data3 = 'EF2_chan3_10mVpp_chan5_gnd';
%
% data1 = 'EF3_chan3_100mVpp';
% data2 = 'EF3_chan3_50mVpp';
% data3 = 'EF3_chan3_10mVpp';
%
% data1 = 'EF4_chan3_100mVpp_chan4_gnd';
% data2 = 'EF4_chan3_50mVpp_chan4_gnd';
% data3 = 'EF4_chan3_10mVpp_chan4_gnd';
%
% data1 = 'EF4_chan4_100mVpp_chan3_gnd';
% data2 = 'EF4_chan4_50mVpp_chan3_gnd';
% data3 = 'EF4_chan4_10mVpp_chan3_gnd';
%
% data1 = 'EF5_chan4_100mVpp';
% data2 = 'EF5_chan4_50mVpp';
% data3 = 'EF5_chan4_10mVpp';
%
% data1 = 'EF6_chan1_100mVpp_chan2_gnd';
% data2 = 'EF6_chan1_50mVpp_chan2_gnd';
% data3 = 'EF6_chan1_10mVpp_chan2_gnd';
%
% data1 = 'EF6_chan2_100mVpp_chan1_gnd';
% data2 = 'EF6_chan2_50mVpp_chan1_gnd';
% data3 = 'EF6_chan2_10mVpp_chan1_gnd';
```

```
39
40
41 a1 = load('channel_data.mat', data1);
42 a2 = load('channel_data.mat', data2);
43 a3 = load('channel_data.mat', data3);
44
45
46 time_values1 = a1.(data1);
47 time_values2 = a2.(data2);
48 time_values3 = a3.(data3);
49
50 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
51 % Compensate for twos compliment
52 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53 for b = [1:1: length(time_values1)];
54
55     if time_values1(b) <= 32767;
56         turned_data1(b) = time_values1(b);
57     else
58         turned_data1(b) = time_values1(b)-65535;
59     end
60 end
61 for b = [1:1: length(time_values2)];
62
63     if time_values2(b) <= 32767;
64         turned_data2(b) = time_values2(b);
65     else
66         turned_data2(b) = time_values2(b)-65535;
67     end
68 end
69 for b = [1:1: length(time_values3)];
70
71     if time_values3(b) <= 32767;
72         turned_data3(b) = time_values3(b);
73     else
74         turned_data3(b) = time_values3(b)-65535;
75     end
76 end
77
78
79 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

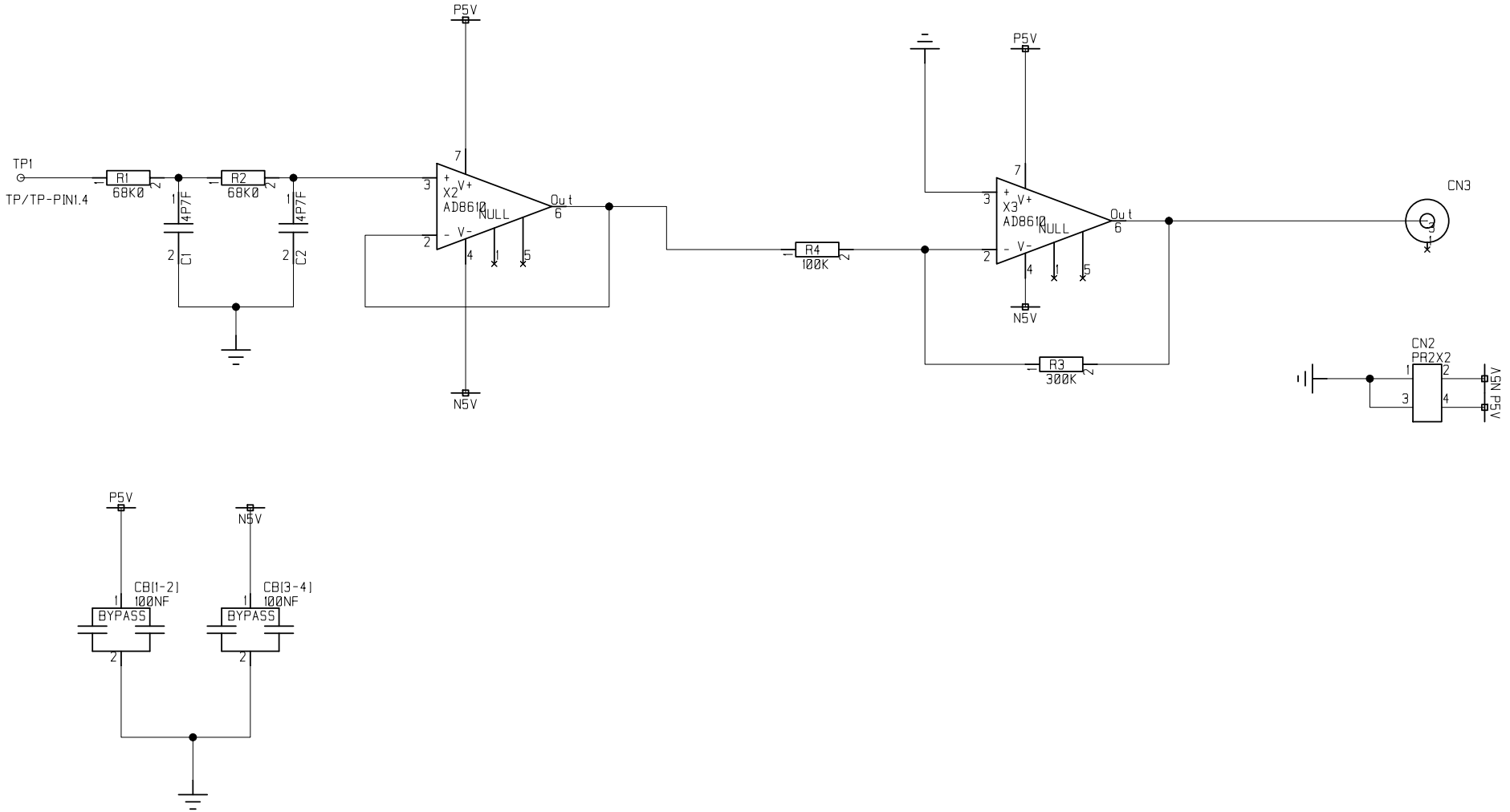
```
%% Remove dropout
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
smoth_data1(1,1) = turned_data1(1,1);
for c = [2:1:(1/2)*length(turned_data1)];
    if abs(smoth_data1(c-1)-turned_data1(c))>500;

        smoth_data1(c) = (2*smoth_data1(c-2)+2*smoth_data1(c-1)+turned_data1(c+1))/5;
    else
        smoth_data1(c) = turned_data1(c);
    end
end
for c = [(1/2)*length(turned_data1):1:length(turned_data1)];
    if abs(smoth_data1(c-1)-turned_data1(c))>2000;

        smoth_data1(c) = (2*smoth_data1(c-2)+2*smoth_data1(c-1)+turned_data1(c+1))/5;
    else
        smoth_data1(c) = turned_data1(c);
    end
end
smoth_data2(1,1) = turned_data2(1,1);
for c = [2:1:(1/2)*length(turned_data2)];
    if abs(smoth_data2(c-1)-turned_data2(c))>500;

        smoth_data2(c) = (2*smoth_data2(c-2)+2*smoth_data2(c-1)+turned_data2(c+1))/5;
    else
        smoth_data2(c) = turned_data2(c);
    end
end
for c = [(1/2)*length(turned_data2):1:length(turned_data2)];
    if abs(smoth_data2(c-1)-turned_data2(c))>1000;

        smoth_data2(c) = (2*smoth_data2(c-2)+2*smoth_data2(c-1)+turned_data2(c+1))/5;
    else
        smoth_data2(c) = turned_data2(c);
    end
```

```
115 end
116 smoth_data3(1,1) = turned_data3(1,1);
117 for c = [2:1:(1/2)*length(turned_data3)];
118     if abs(smoth_data3(c-1)-turned_data3(c))>200;
119
120         smoth_data3(c) = (2*smoth_data3(c-2)+2*
                smoth_data3(c-1)+turned_data3(c+1))/5;
121     else
122         smoth_data3(c) = turned_data3(c);
123     end
124 end
125 for c = [(1/2)*length(turned_data3):1:length(
        turned_data3)];
126     if abs(smoth_data3(c-1)-turned_data3(c))>700;
127
128         smoth_data3(c) = (2*smoth_data3(c-2)+2*
                smoth_data3(c-1)+turned_data3(c+1))/5;
129     else
130         smoth_data3(c) = turned_data3(c);
131     end
132 end
133
134
135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
136 %% Plot
137 Fs = 2893.5*6;  % Readout frequency
138 T = 1/Fs;       % Timestep
139 L1 = length(smoth_data1); % Length of data set
140 t1 = (0:L1-1)*T; % Time vector
141 L2 = length(smoth_data2); % Length of data set
142 t2 = (0:L2-1)*T; % Time vector
143 L3 = length(smoth_data3); % Length of data set
144 t3 = (0:L3-1)*T; % Time vector
145 %plot(t1,smoth_data1);
146
147
148 timeplot = plot(t1,smoth_data1,t2, smoth_data2, t3,
        smoth_data3);
149 %timeplot = plot(t1,turned_data1,t2, turned_data2, t3,
        turned_data3);
150 %axis([6.5,7.5,-5000,5000]);
```

```
151 %setleg =legend(data1, data2, data3)
152 setleg =legend('Kanal 1 100 mV','Kanal 1 50 mV', 'Kanal
        1 10 mV')
153 grid 'on';
154 set(gca, 'LineWidth', 2, 'FontSize', 16);
155 set(setleg, 'Interpreter','none');
156 xlabel('Tid [s]');
157 ylabel('Amplitude [Value]');
```

Tillegg B

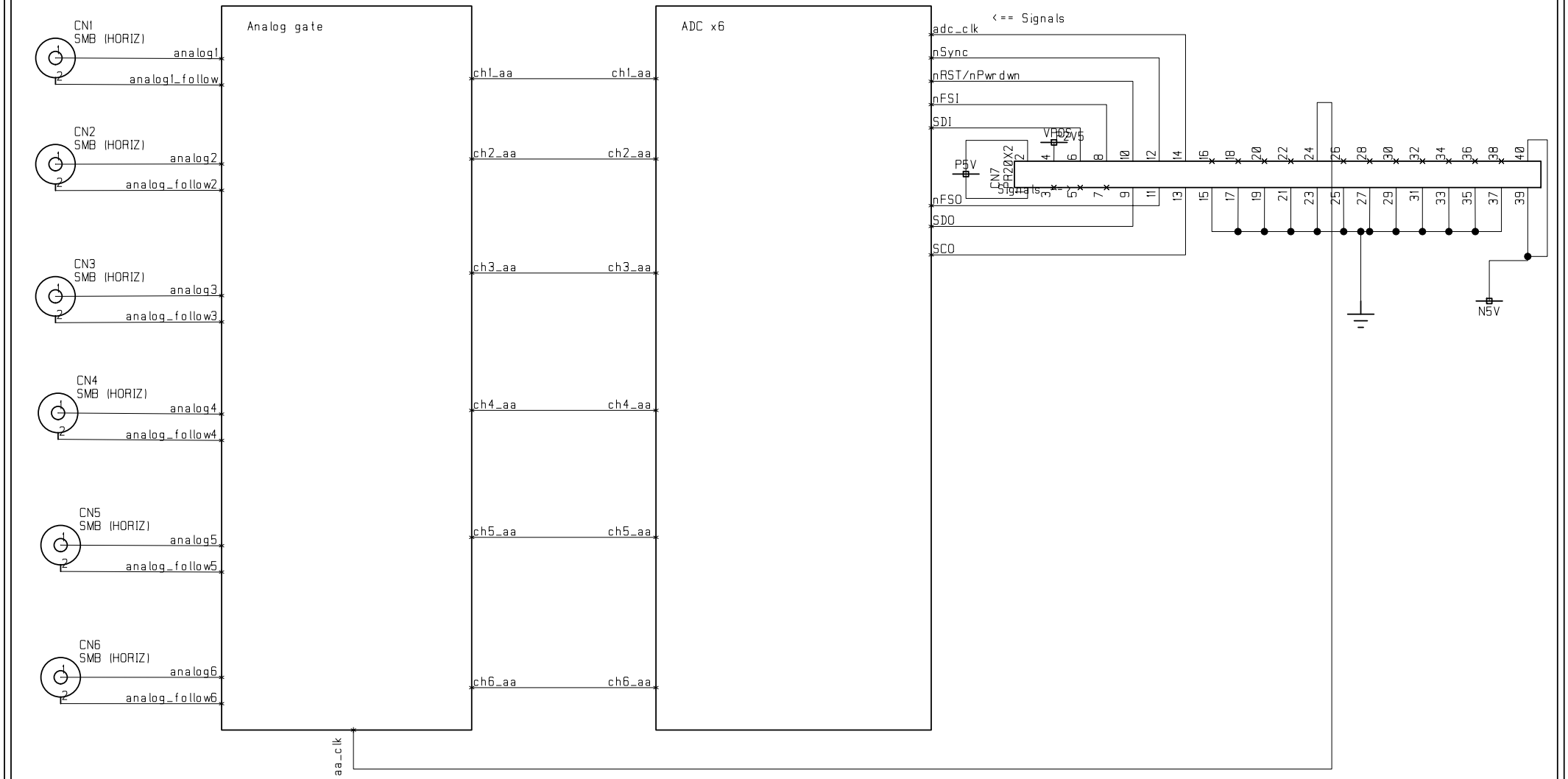
Skjema Probekort



PROJECT:	E-felt Probekort
DRAWING:	
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 1:1	(c) copyright. All rights reserved.

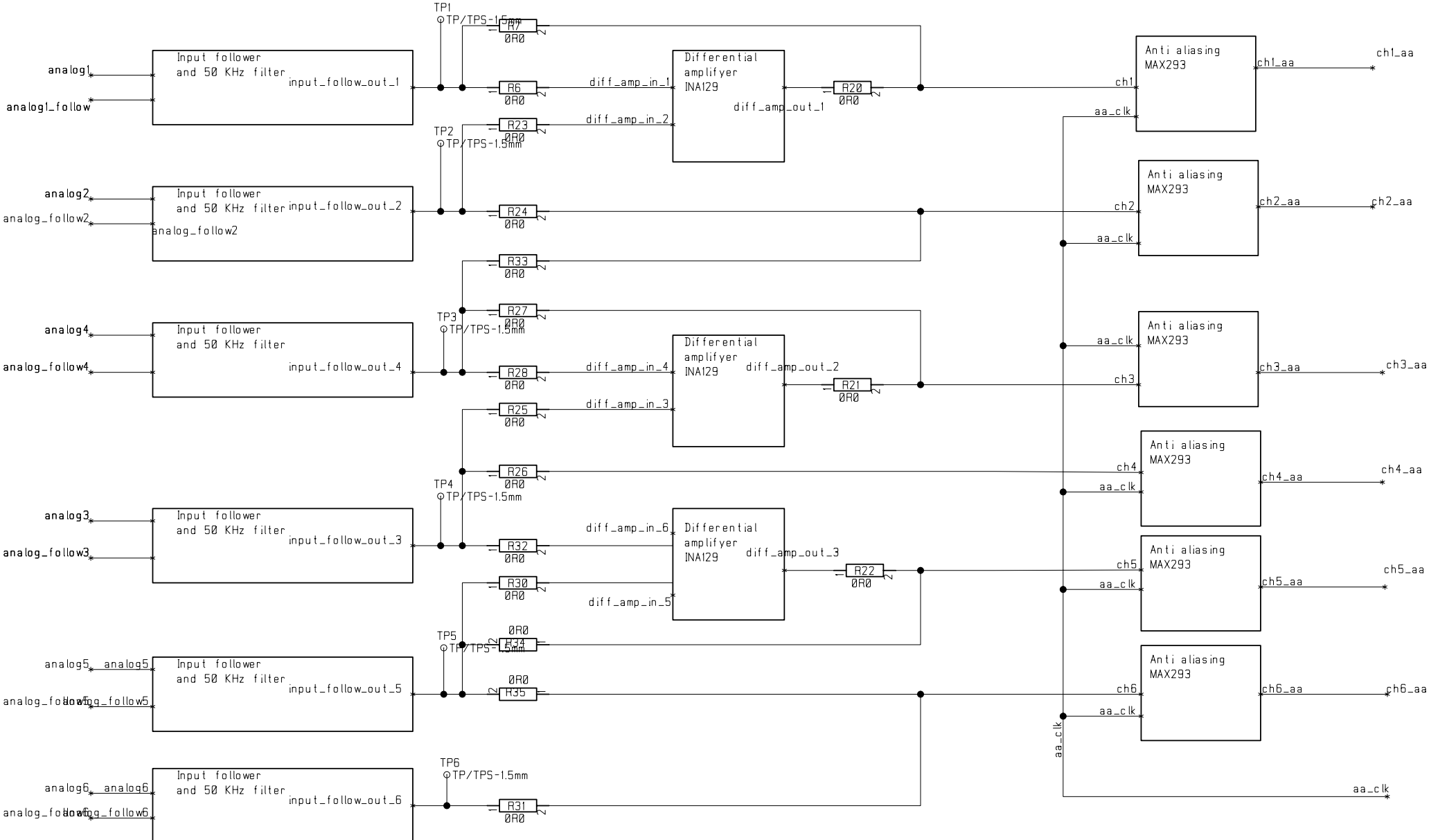
Tillegg C

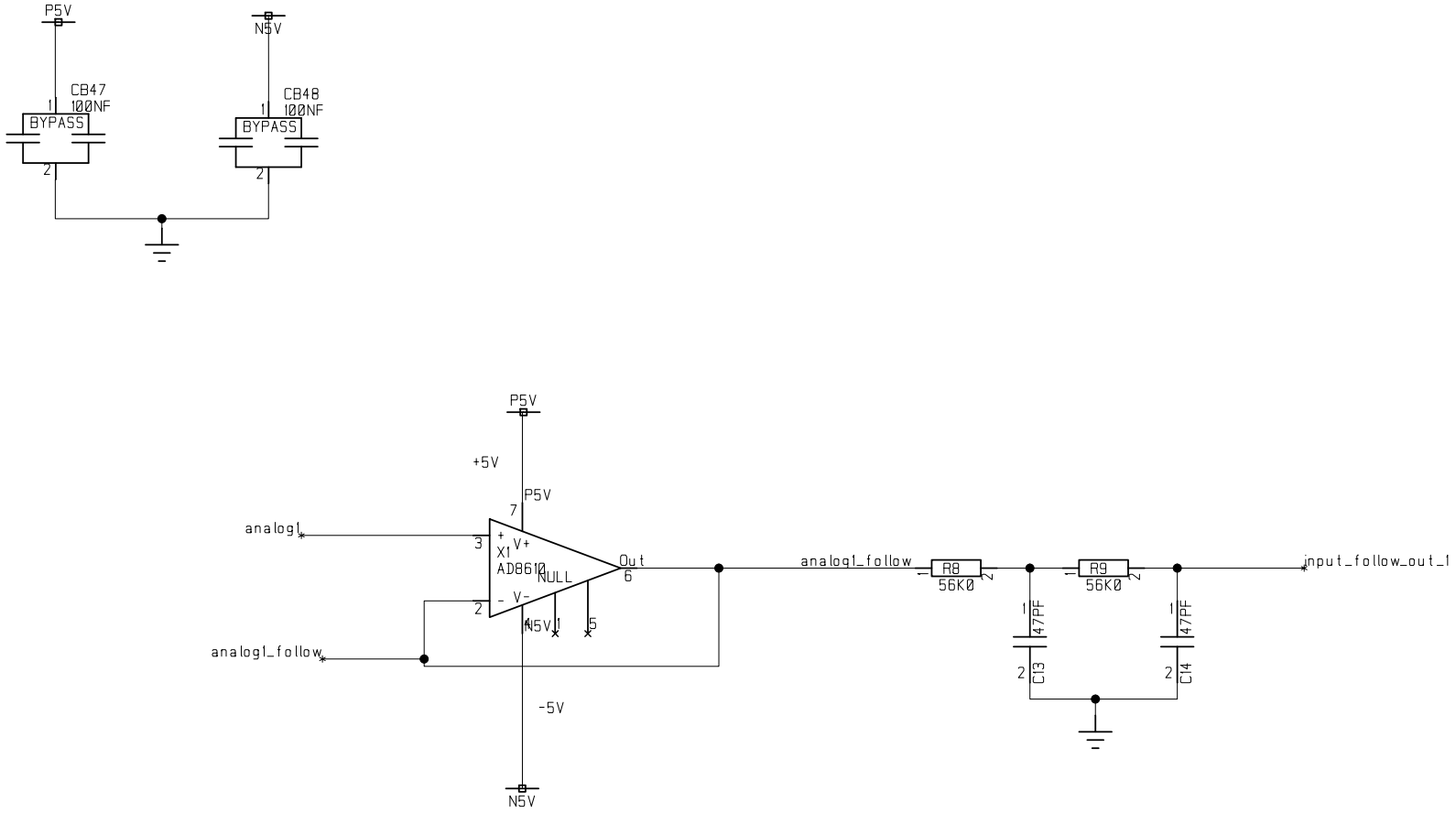
Skjema Analogkort



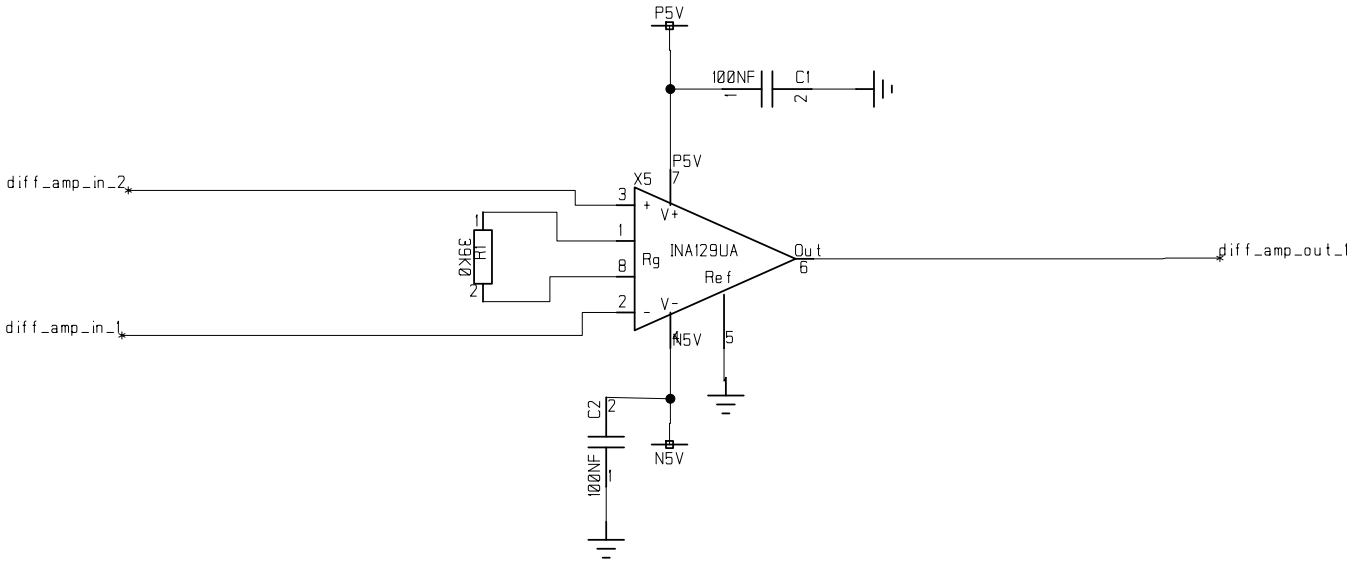
SH1
PC104 MARK
TP/SH-PC104-MARK
SH2
PC104 MARK
TP/SH-PC104-MARK
SH3
PC104 MARK
TP/SH-PC104-MARK
SH4
PC104 MARK
TP/SH-PC104-MARK

PROJECT:	E-FELT ANALOGKORT
DRAWING:	Top
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 1:31	(c) copyright. All rights reserved.

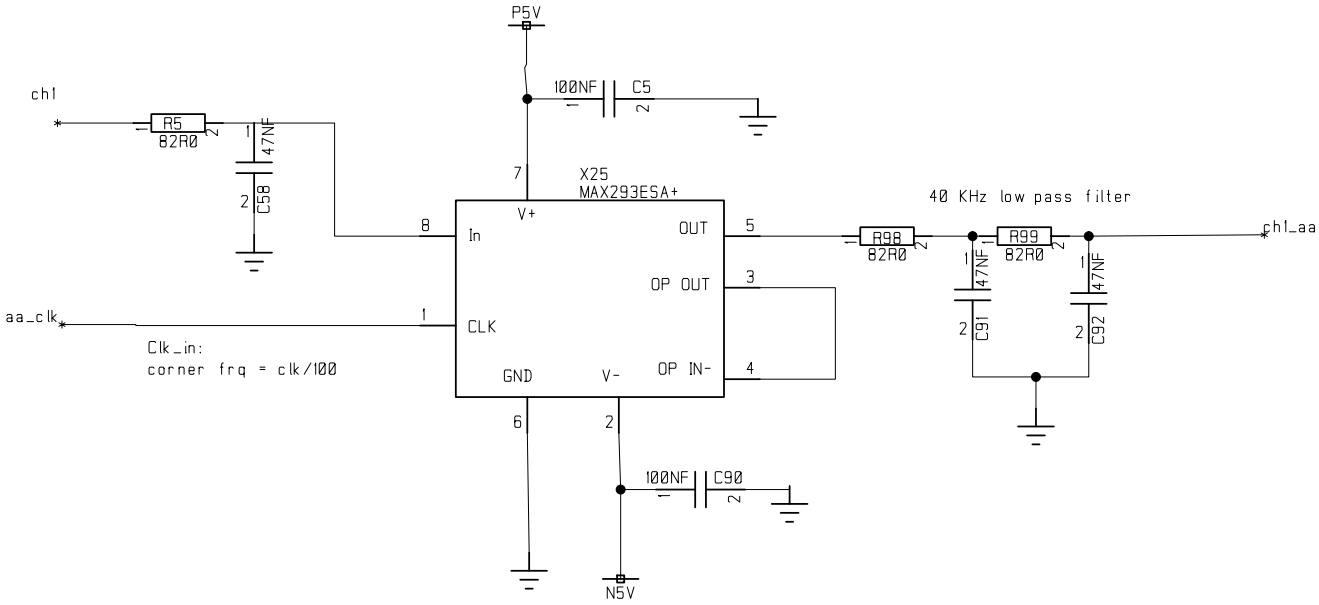

TP1
TP/TPS-1.5mm
TP2
TP/TPS-1.5mm
TP3
TP/TPS-1.5mm
TP4
TP/TPS-1.5mm
TP5
TP/TPS-1.5mm
TP6
TP/TPS-1.5mm
analog1
analog1_follow
analog2
analog_follow2
analog3
analog_follow3
analog4
analog_follow4
analog5
analog_follow5
analog6
analog_follow6
Input follower
and 50 KHz filter
input_follow_out_1
input_follow_out_2
input_follow_out_3
input_follow_out_4
input_follow_out_5
input_follow_out_6
analog_follow2
R7
R6
R23
R24
R33
R27
R28
R25
R26
R32
R30
R34
R35
R31
0R0
Differential
amplifyer
INA129
diff_amp_in_1
diff_amp_in_2
diff_amp_out_1
diff_amp_in_4
diff_amp_in_3
diff_amp_out_2
diff_amp_in_6
diff_amp_in_5
diff_amp_out_3
R20
R21
R22
Anti aliasing
MAX293
ch1
ch2
ch3
ch4
ch5
ch6
ch1_aa
ch2_aa
ch3_aa
ch4_aa
ch5_aa
ch6_aa
aa_clk
Koblingsnettverk
PROJECT:
E-felt Analogkort
DRAWING:
Analog Gate
DATE: 3.9.12
VER: 3
SHEET: 2:31
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.



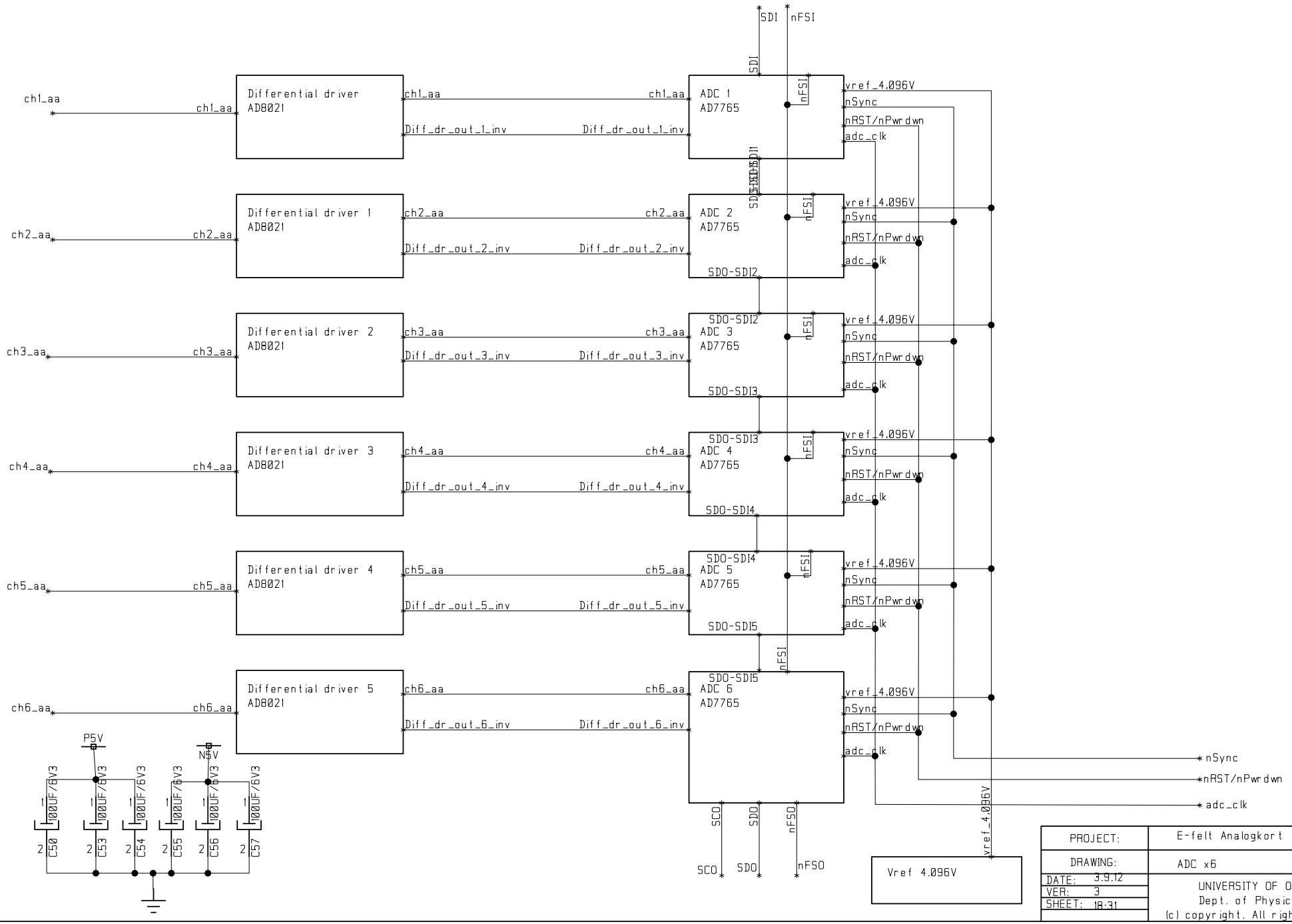
PROJECT:	E-felt Analogkort
DRAWING:	Input follower
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 3:31	(c) copyright. All rights reserved.

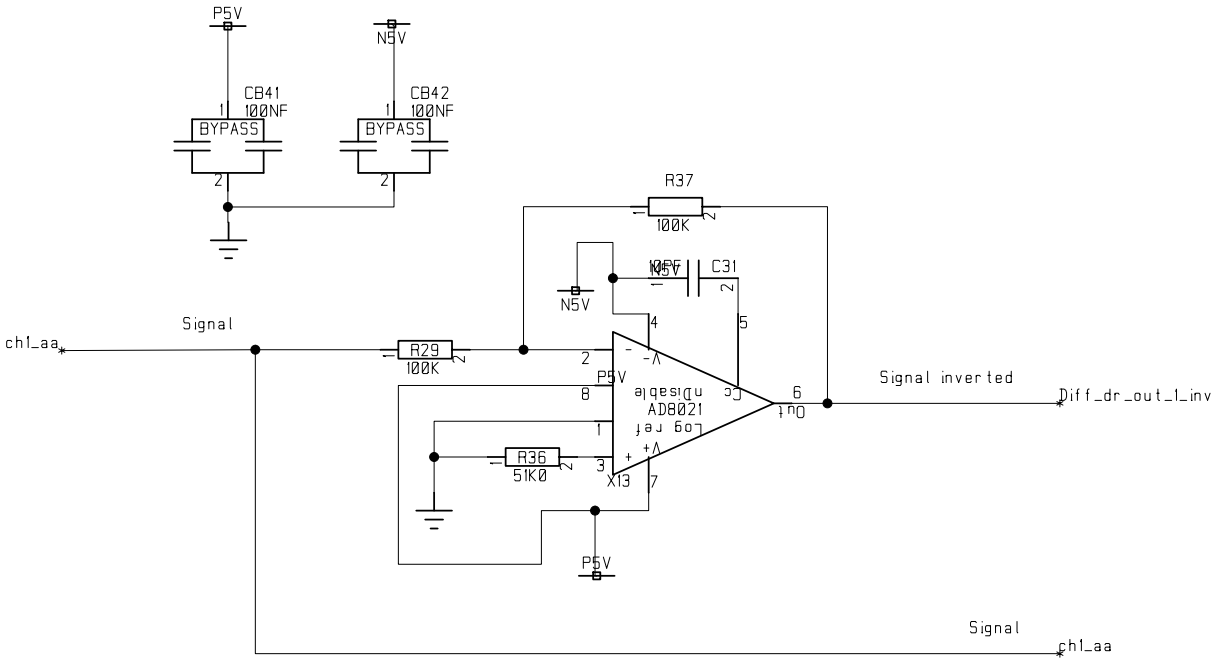


PROJECT:	E-felt Analogkort
DRAWING:	Differential Amplifier
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 9:31	

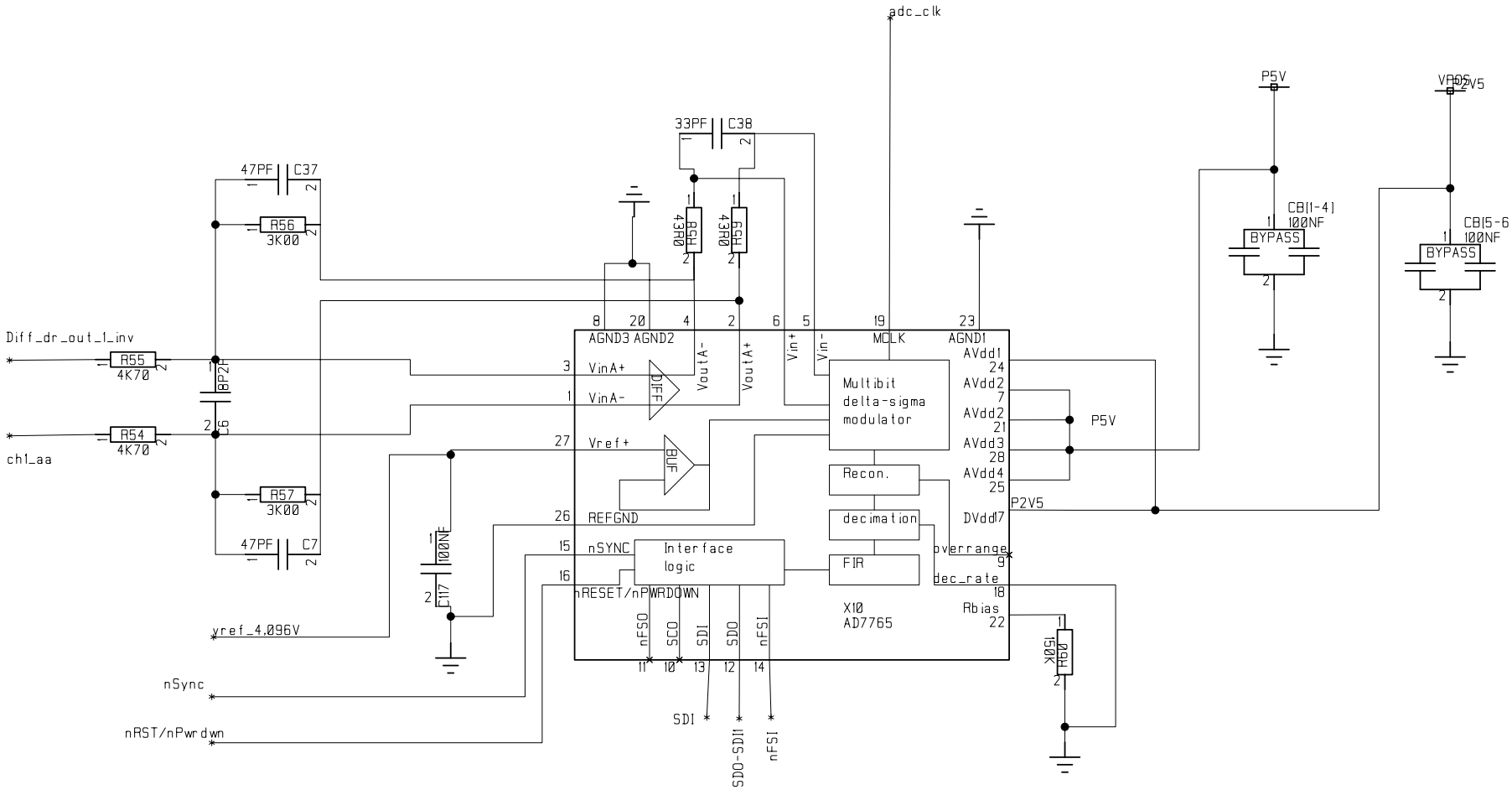


PROJECT:	E-felt Analogkort
DRAWING:	Antialiasing
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 12:31	(c) copyright. All rights reserved.

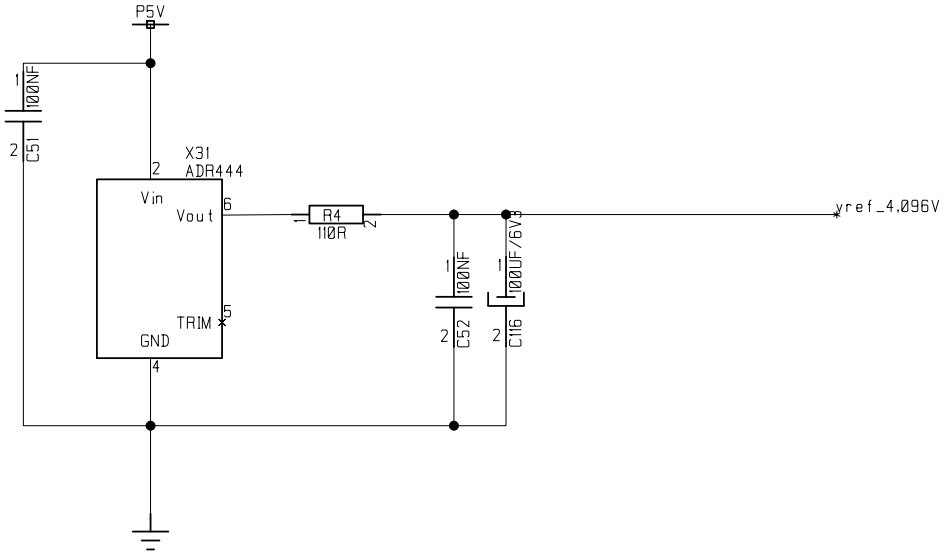

SDI
nFSI
ch1_aa
Differential driver
AD8021
ch1_aa
Diff_dr_out_1_inv
ADC 1
AD7765
vref_4.096V
nSync
nRST/nPwrdwn
adc_clk
SDO-SDI1
ch2_aa
Differential driver 1
AD8021
Diff_dr_out_2_inv
ADC 2
AD7765
SDO-SDI2
ch3_aa
Differential driver 2
AD8021
Diff_dr_out_3_inv
ADC 3
AD7765
SDO-SDI3
ch4_aa
Differential driver 3
AD8021
Diff_dr_out_4_inv
ADC 4
AD7765
SDO-SDI4
ch5_aa
Differential driver 4
AD8021
Diff_dr_out_5_inv
ADC 5
AD7765
SDO-SDI5
ch6_aa
Differential driver 5
AD8021
Diff_dr_out_6_inv
ADC 6
AD7765
SCO
SDO
nFSO
P5V
N5V
100UF/6V3
C50
C53
C54
C55
C56
C57
Vref 4.096V
PROJECT:
E-felt Analogkort
DRAWING:
ADC x6
DATE: 3.9.12
VER: 3
SHEET: 18:31
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.



PROJECT:	E-felt Analogkort
DRAWING:	Differential driver
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 21:31	(c) copyright. All rights reserved.



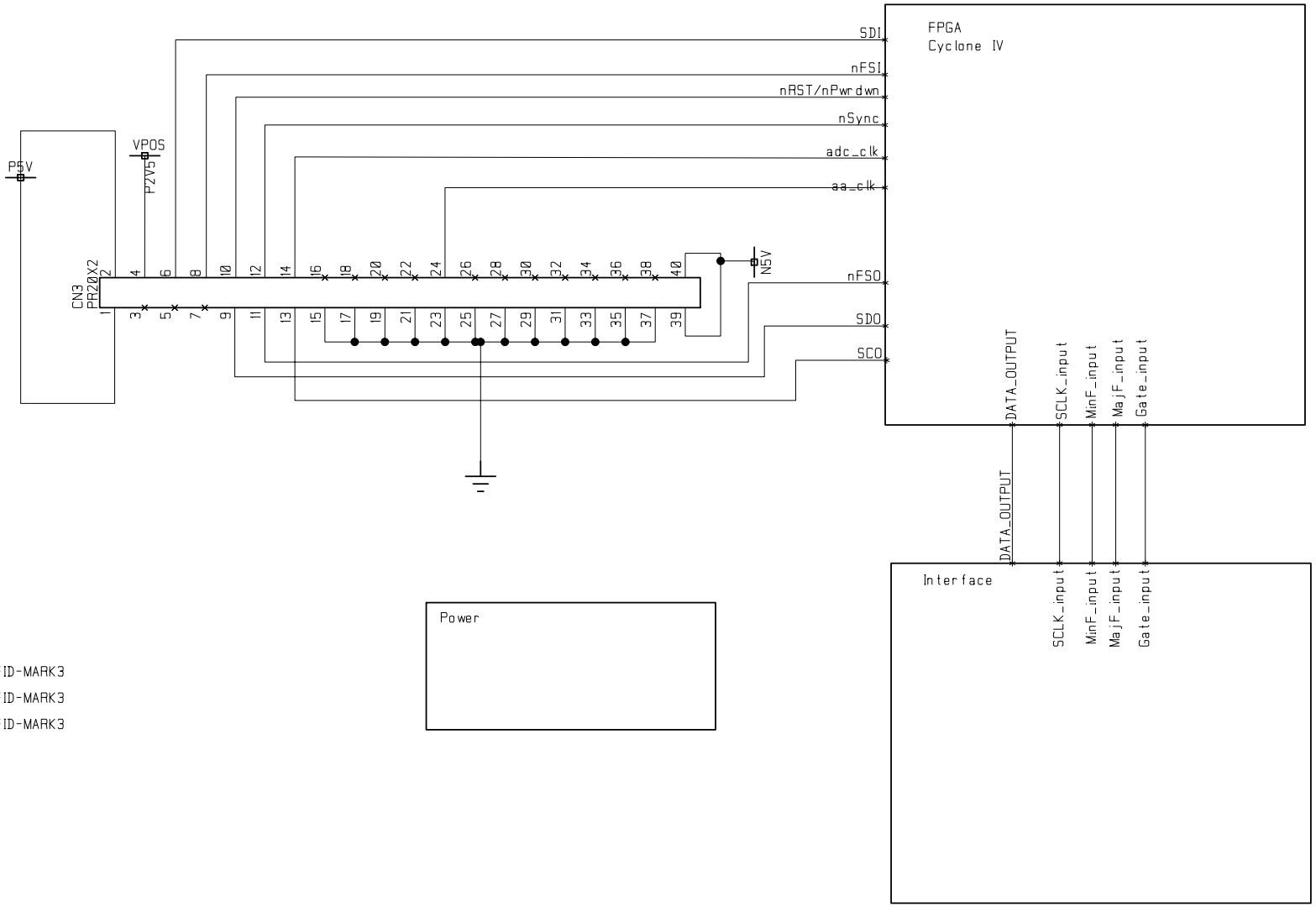
PROJECT:	E-felt Analogkort
DRAWING:	ADC AD7765
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 25:31	



PROJECT:	E-felt Analogkort
DRAWING:	Referansespenning 4.096V
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 31:31	

Tillegg D

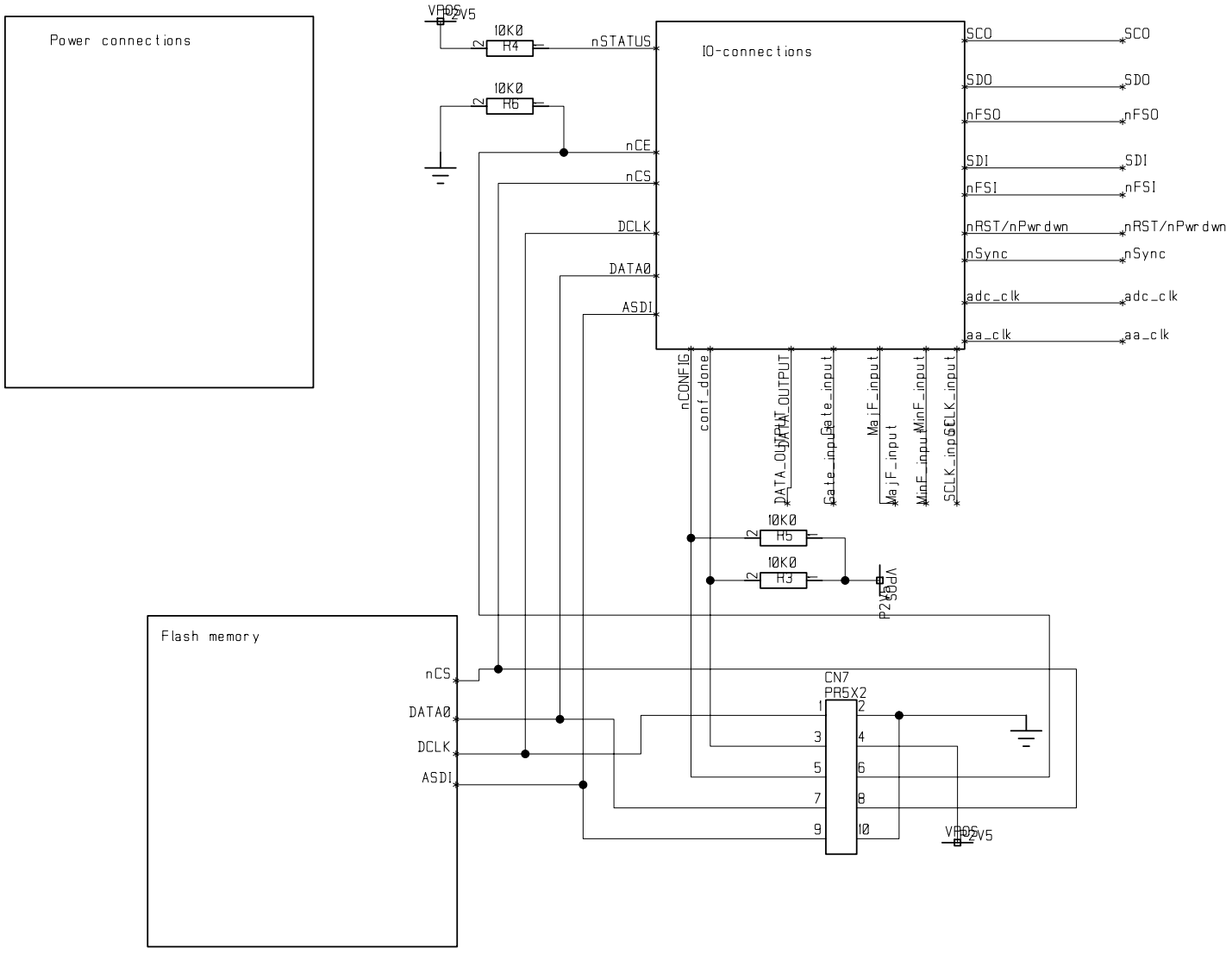
Skjema Digitalkort



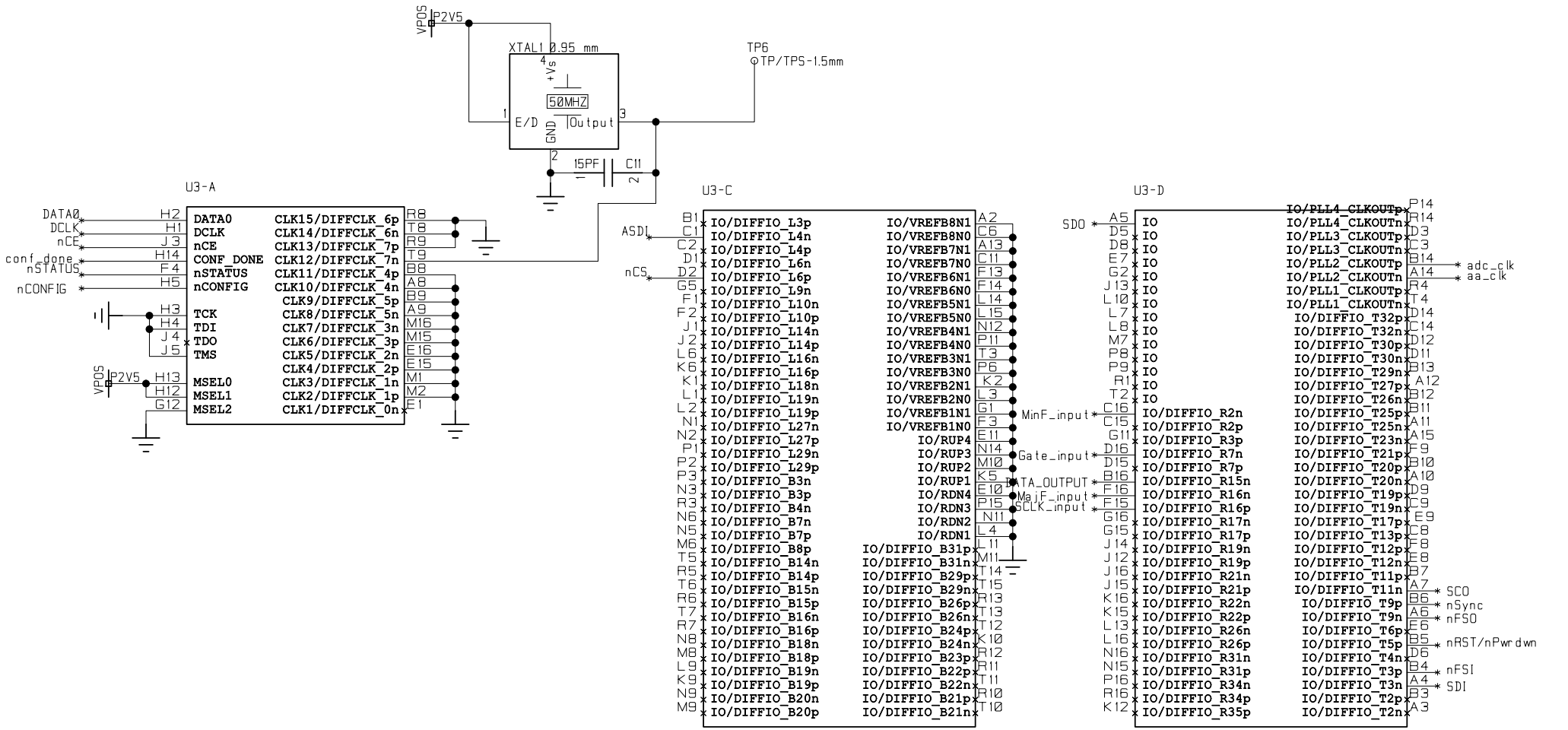
MARK1
⊗TP/FID-MARK3
MARK2
⊗TP/FID-MARK3
MARK3
⊗TP/FID-MARK3

T4
⊗PC104_MARK
SH/PC104-MARK
T1
⊗PC104_MARK
SH/PC104-MARK
T2
⊗PC104_MARK
SH/PC104-MARK
T3
⊗PC104_MARK
SH/PC104-MARK

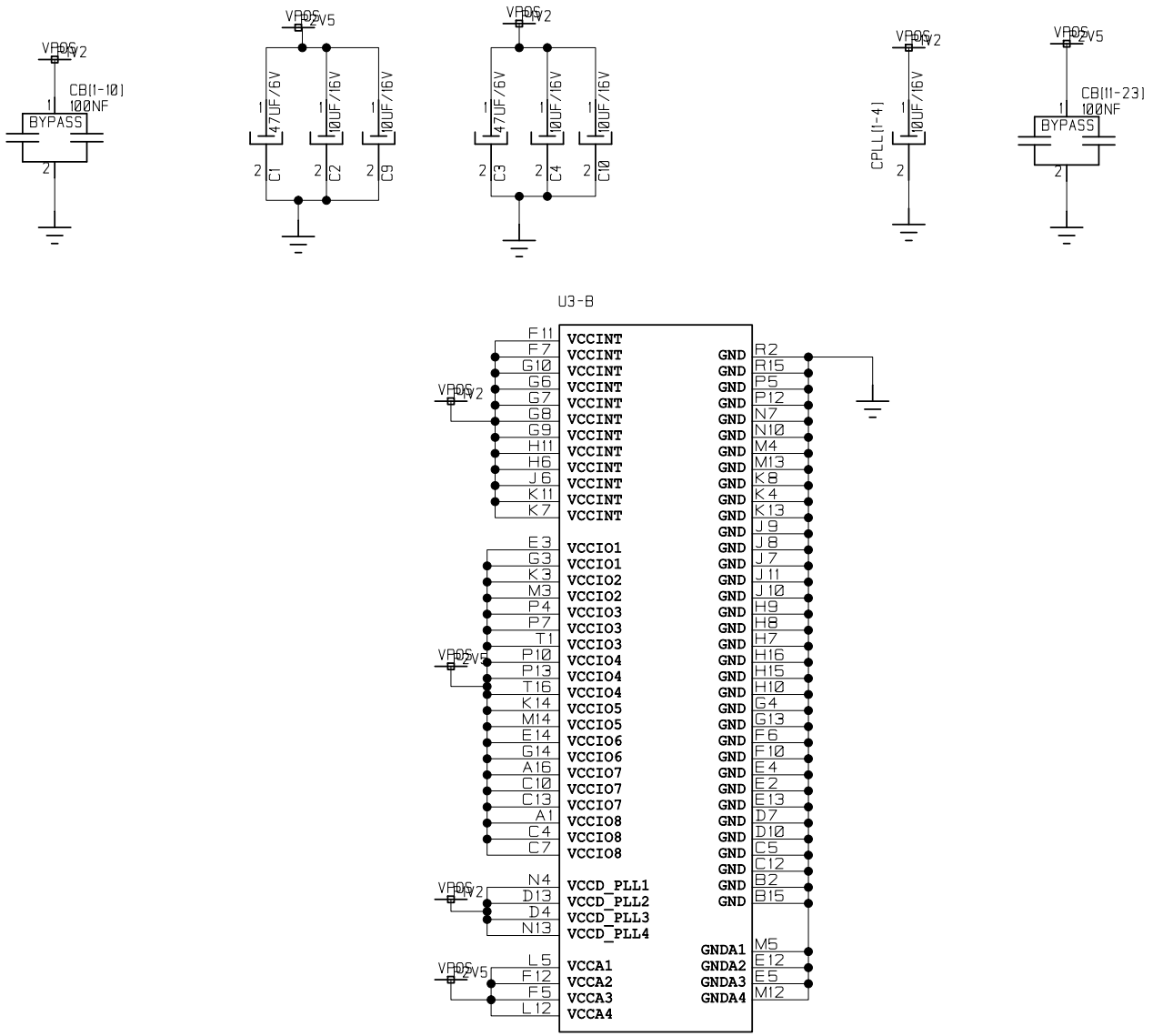
PROJECT:	E-felt Digitalkort
DRAWING:	Top
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 1:9	



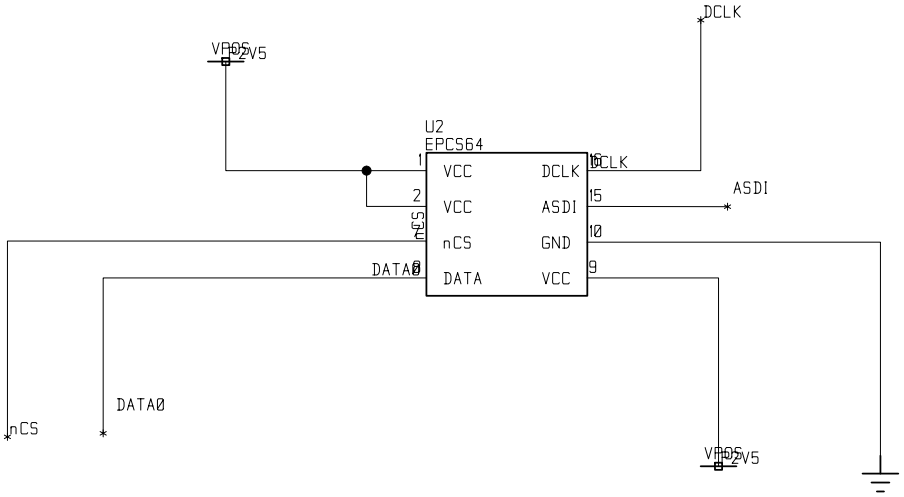
PROJECT:	E-felt digitalkort
DRAWING:	FPGA
DATE: 3.9.12 VER: 3 SHEET: 2:9	UNIVERSITY OF OSLO Dept. of Physics

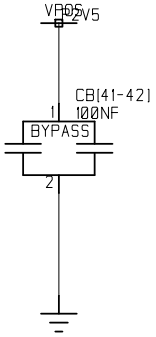


PROJECT:	E-felt digitalkort
DRAWING:	IO-connections
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 3:9	(c) copyright. All rights reserved.

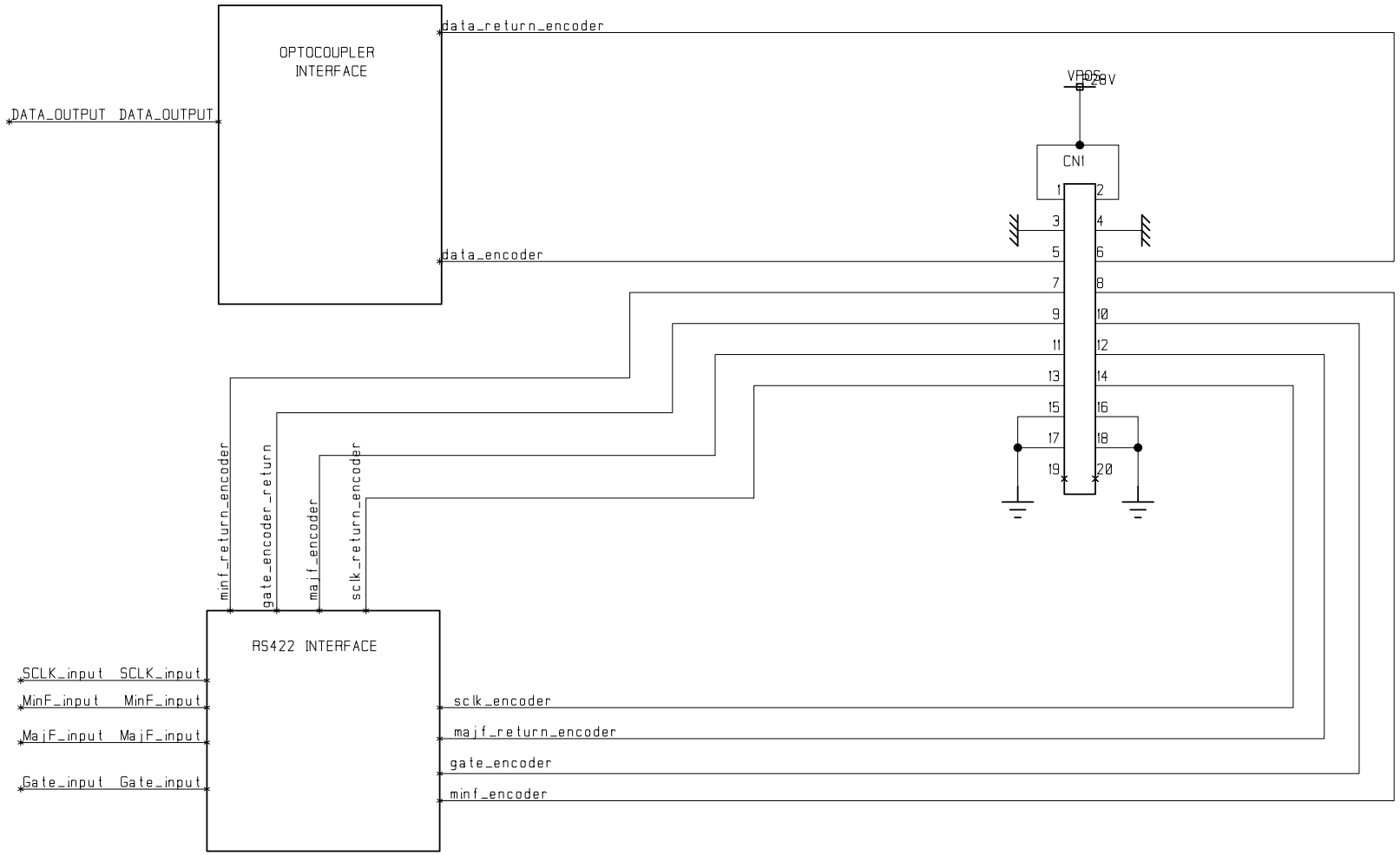


PROJECT:	E-felt digitalkort
DRAWING:	Power connections
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 4:9	

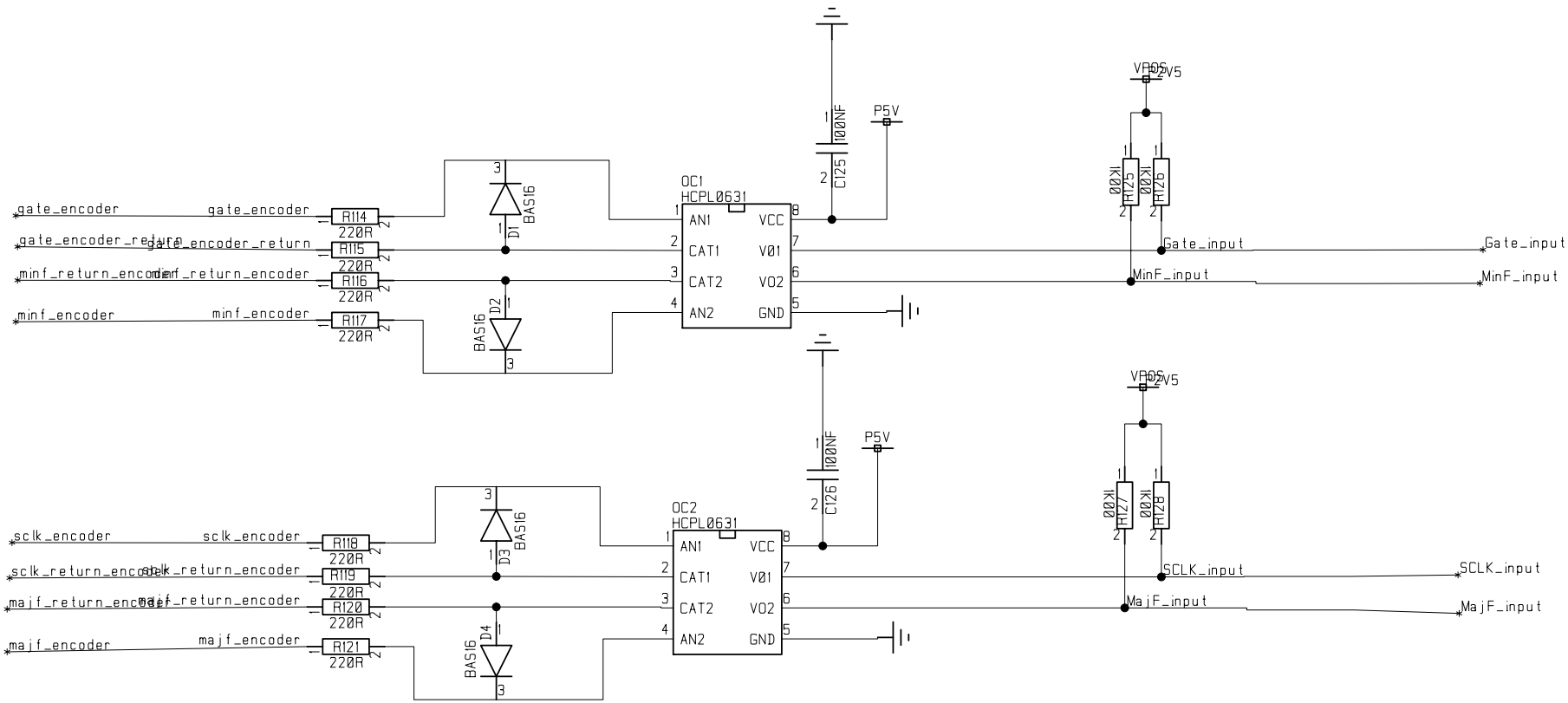




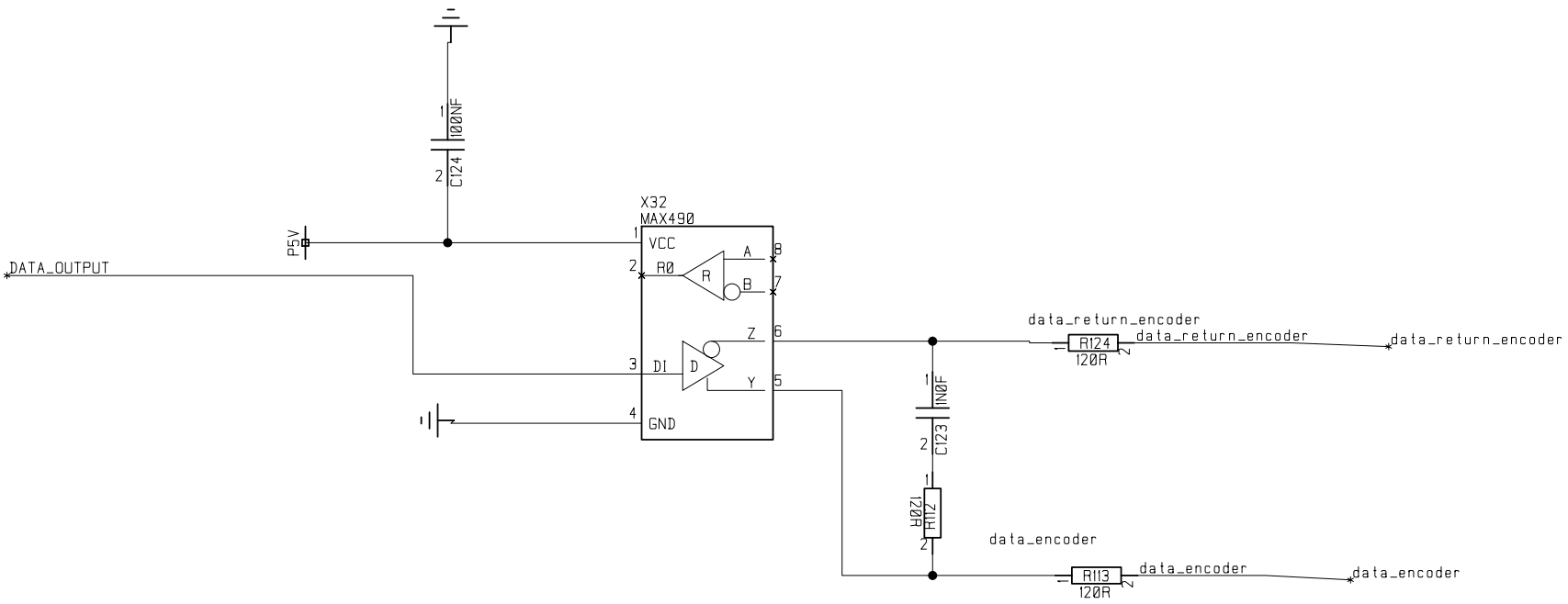
PROJECT:	E-felt digitalkort
DRAWING:	Flash memory
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 5:9	



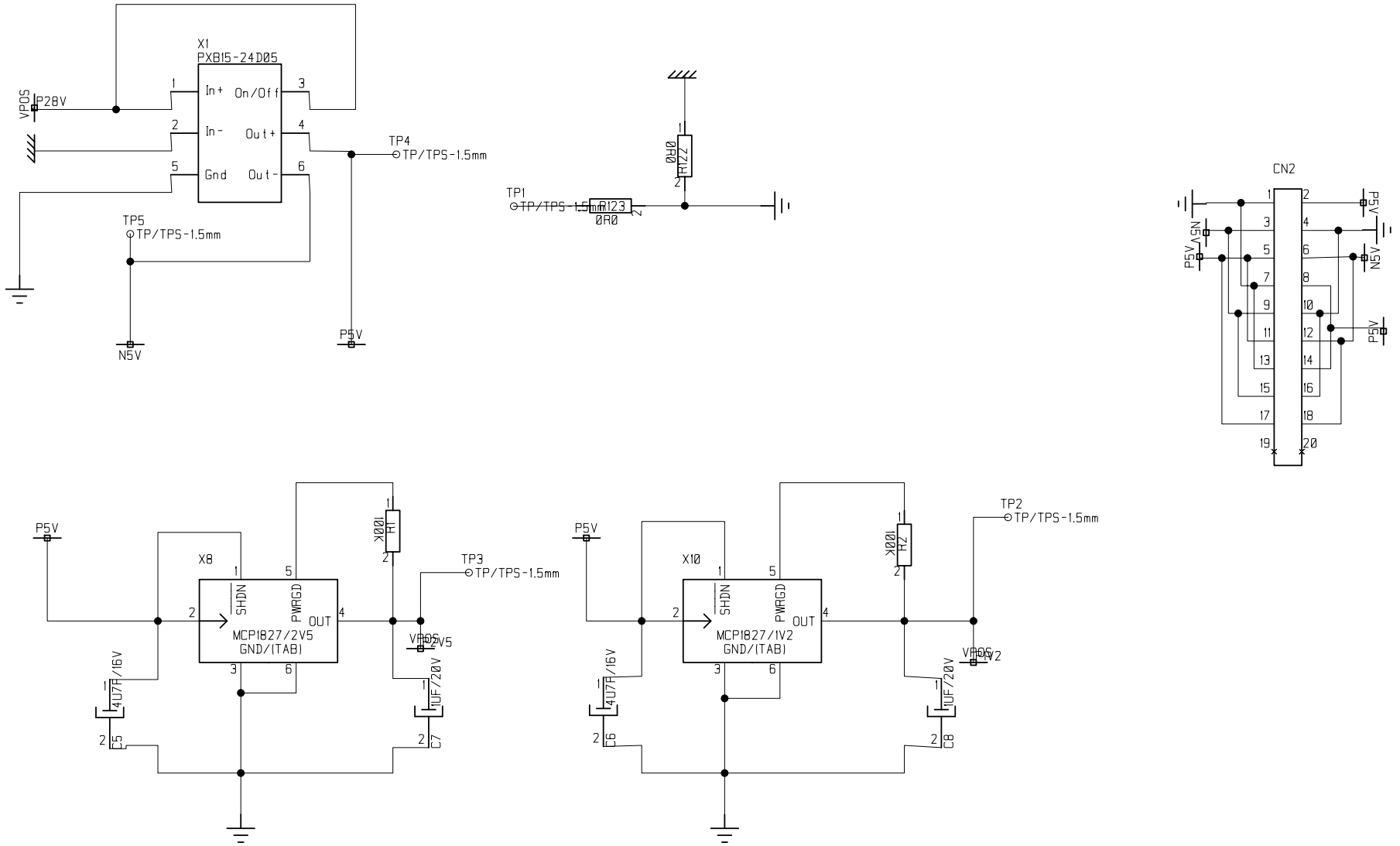
PROJECT:	Efelt digitalkort interface
DRAWING:	
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 6:9	



PROJECT:	E-felt digitalkort
DRAWING:	Optocouplers
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 7:9	



PROJECT:	E-felt digitalkort
DRAWING:	Line Driver
DATE: 3.9.12 VER: 3 SHEET: 8:9	UNIVERSITY OF OSLO Dept. of Physics

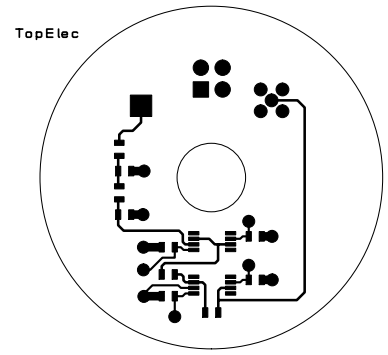


PROJECT:	E-felt digitalkort
DRAWING:	Power
DATE: 3.9.12	UNIVERSITY OF OSLO
VER: 3	Dept. of Physics
SHEET: 9:9	(c) copyright. All rights reserved.

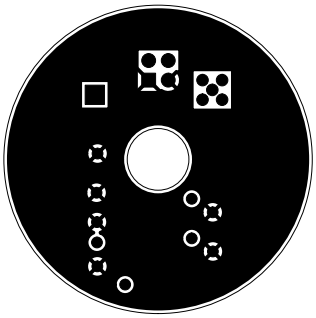
Tillegg E

Utlegg på PCB

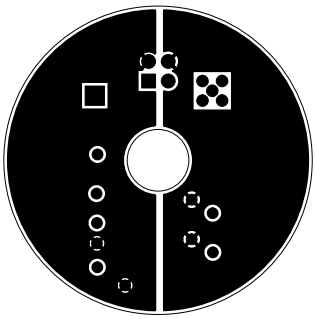
Probekort



(a) Topplaget på probekortet



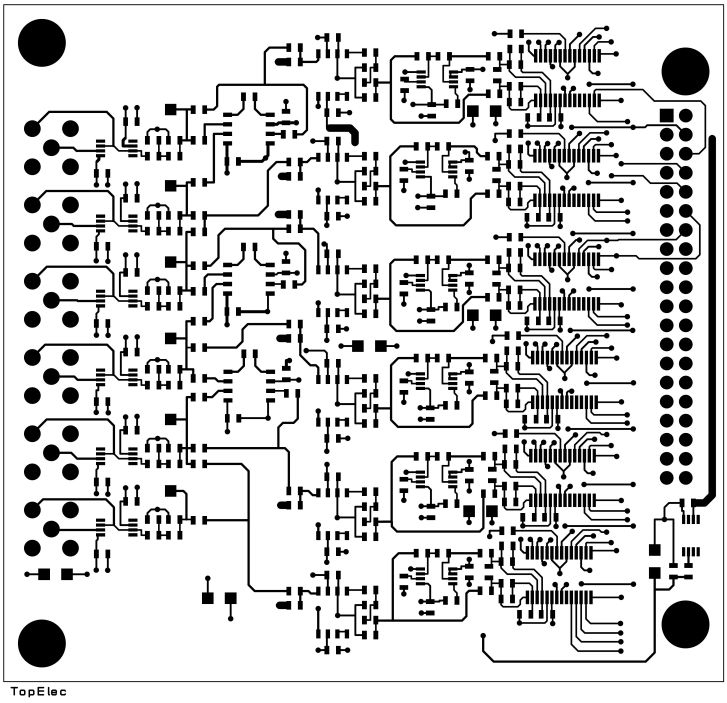
(b) Jordlaget på probekortet



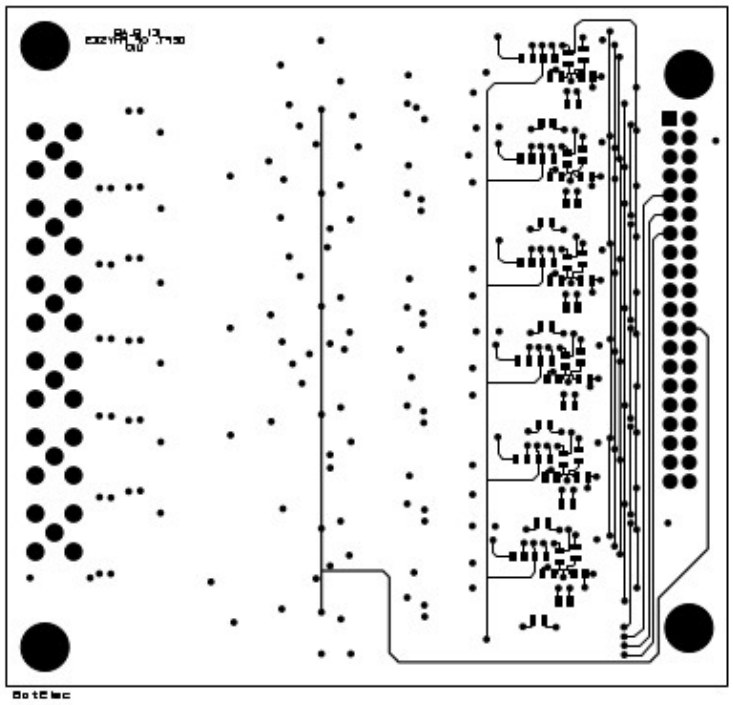
(c) Powerlaget til probekortet

Figur E.1: Utlegg av probekortet.

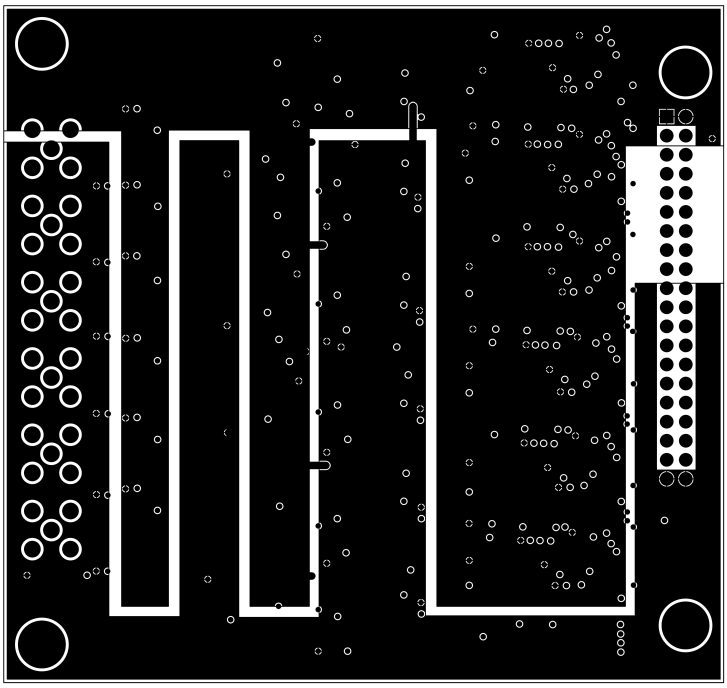
Analogkort



Figur E.2: Topplag på analogkortet

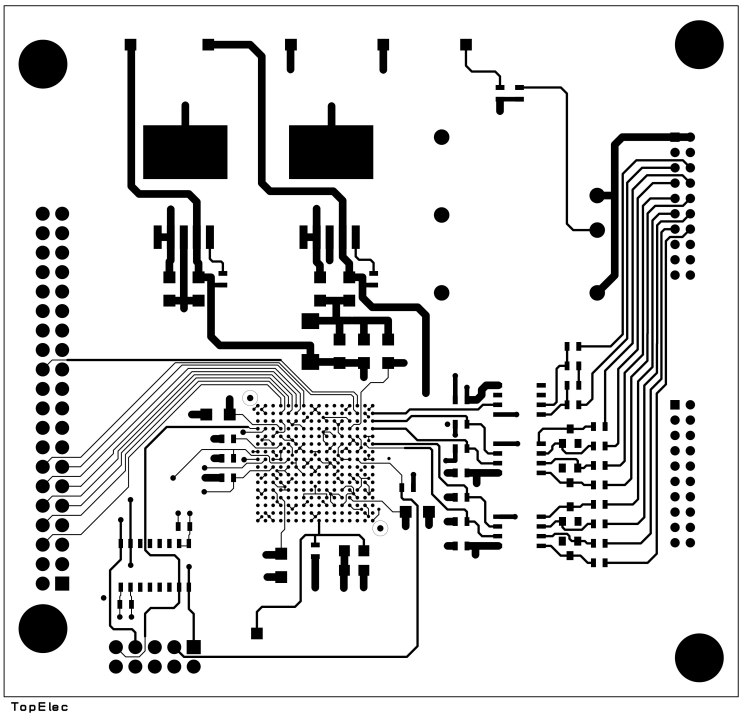


Figur E.3: Bunnlaget på analogkortet

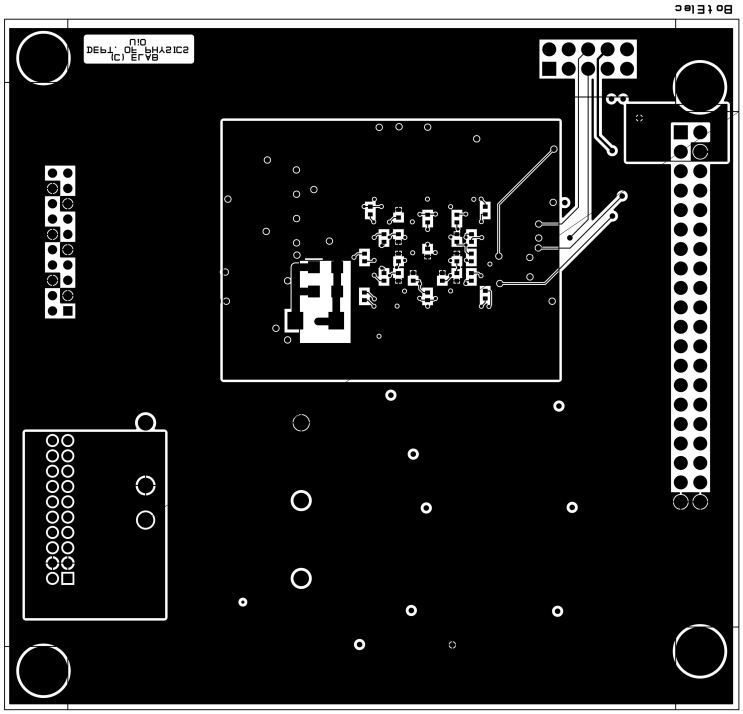


Figur E.4: Powerlaget til analogkortet

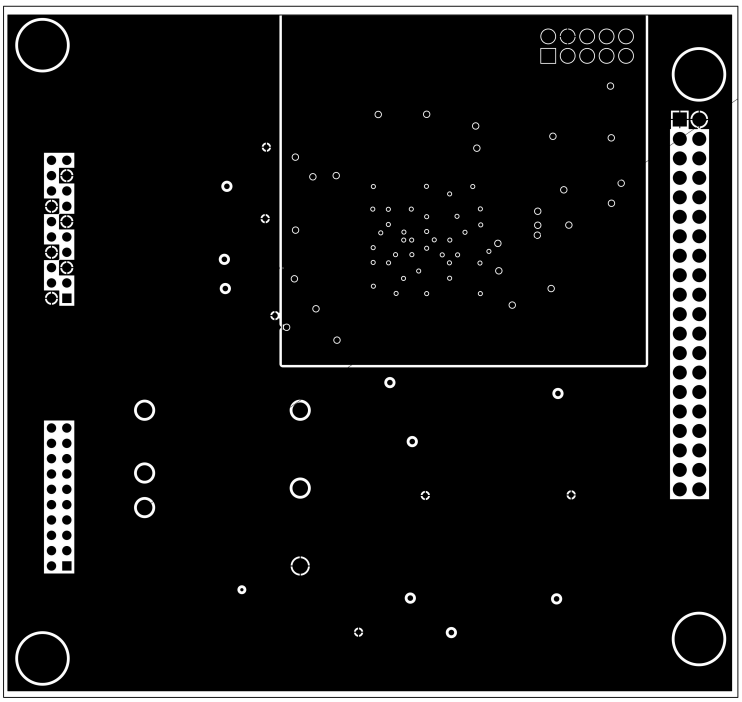
Digitalkort



Figur E.5: Topplaget til digitalkortet



Figur E.6: Bunnlaget til digitalkortet



Figur E.7: Powerlaget til digitalkortet

Tillegg F

VHDL-kode

top.vhd

```
1  ----------------------------------------------------------------
2  --  Name:               Top
3  --  Writer:             Elling Hauge-Iversen
4  --  Date:               21.11.2012
5  --  Ver:                1
6  --  Description:        Top design of efeltkort
7  ----------------------------------------------------------------
8
9  library IEEE;
10 use IEEE.std_logic_1164.all;
11 use IEEE.numeric_std.all;
12
13 entity top is
14   port  ( clk_50MHz      : in std_logic;
15           -- output to anti aliasing filter
16           --aa_clock       : out std_logic;
17           -- lines to/from adc
18           adc_clk        : out std_logic;
19           nReset_nPwdwn  : out std_logic;
20           SCO            : in std_logic; -- clock from adc
21           nFSO           : in std_logic; -- frame sync from adc
22           SDO            : in std_logic; -- data from adc
23           nFSI           : out std_logic; -- frame sync to adc
24           --SDI             : out std_logic; -- data to adc
25           nSync          : out std_logic; -- sync adc
26           -- interface
27           not_Sclk       : in std_logic; -- system clock
28           not_Gate       : in std_logic;
29           not_MajF       : in std_logic; -- Major frame, not used
30           not_MinF       : in std_logic; -- Minor frame
31           Data           : out std_logic -- data output
```

```
32          );
33 end entity;
34
35
36
37
38
39
40
41 architecture str of top is
42
43 -- constant declaration
44 constant number_of_adc         : natural := 6;
45 constant regwidth              : natural := 32;
46 constant data_out_width        : natural := 16;
47 constant fifo_count_bit_width  : natural := 11;
48 constant fifo_word_length      : natural := 2048;
49
50
51 -- signal declaration
52 signal sys_clk                 : std_logic;
53 signal sys_rst                 : std_logic;
54 signal FIFO_clr                : std_logic := '0'; -- fifo clear
55 signal FIFO_read_req           : std_logic;
56 signal FIFO_write_req          : std_logic;
57 signal read_enable             : std_logic;
58 signal FIFO_full               : std_logic;
59 signal FIFO_empty              : std_logic;
60 signal read_in_vec             : std_logic_vector(15 downto 0);
61 signal read_out_vec            : std_logic_vector(15 downto 0);
62 signal data_out                : std_LOGIC;
63 signal int_aa_clk              : std_LOGIC;
64 signal value_to_send           : std_logic_vector(data_out_width-1
       downto 0); -- value from FIFO to interface
65 signal get_new_value           : std_logic; -- Value request from
       interface
66 signal usedw                   : std_logic_vector(
       fifo_count_bit_width-1 downto 0); -- Used space in FIFO
67 signal clk_ok                 : std_logic;
68 signal read_ok                 : std_logic;
69 signal MCLK                    : std_logic;
70
71
72 signal test              : std_LOGIC_VECTOR(data_out_width-1 downto
       0);
73
74
75
76
```

```
77
78
79
80 ------------------------------------------
81 -- component declaration
82 ------------------------------------------
83   -- clk manager
84
85   component clk_div IS
86   PORT
87   (
88     inclk0     : IN STD_LOGIC  := '0'; --input clock
89     c0         : OUT STD_LOGIC ;-- ADC clock 9.5 MHz
90     --c1          : OUT STD_LOGIC ;-- anti aliasing clock 500 KHz
91     locked     : OUT STD_LOGIC
92   );
93 end component;
94
95   -- FIFO
96   component Fifo_16bit IS
97   PORT
98   (
99     clock    : IN STD_LOGIC ;
100    data     : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
101    rdreq    : IN STD_LOGIC ;
102    sclr     : IN STD_LOGIC ;
103    wrreq    : IN STD_LOGIC ;
104    empty    : OUT STD_LOGIC ;
105    full     : OUT STD_LOGIC ;
106    q        : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
107    usedw    : OUT STD_LOGIC_VECTOR (fifo_count_bit_width-1 DOWNTO
         0)
108  );
109 end component;
110
111  -- read_adc
112  component read is
113  generic ( number_of_adc : natural := 1;
114            regwidth      : natural := 32);
115  port (  clk             : in std_logic;
116          rst             : in std_logic;
117          nFSO            : in std_logic;
118          SDO             : in std_logic;
119          Rcontr          : in std_logic; -- enable read
120          SCO             : in std_logic;
121          data_vec        : out std_logic_vector(15 downto 0);
122          datavalid       : out std_logic;
123          read_ok         : out std_logic
124        );
```

```
125
126   end component;
127
128
129
130
131
132 -- Control
133
134 component controller is
135   generic ( fifo_count_bit_width  : natural := 2;
136             fifo_word_length      : natural := 200 );
137
138   port (    clk           : in std_logic;
139             rst           : in std_logic;
140             Fifo_empty    : in std_logic;
141             Fifo_full     : in std_logic;
142             Wcount        : in std_logic_vector(
                  fifo_count_bit_width-1 downto 0);
143             nsync         : out std_logic;
144             nReset        : out std_logic;
145             read_enable : out std_logic
146         );
147 end component;
148
149   -- Interface
150 component interface_sonderakett is
151   generic (   data_out_width             : integer := 16;
152               First_valid_word_in_frame  : integer := 1;
153               Valid_word_increment       : integer   := 2;
154               fifo_count_bit_width       : integer);
155
156   port  (
157     clk           : in std_logic; -- internal clock
158     rst           : in std_logic; -- reset
159     value_to_send : in std_logic_vector(data_out_width-1 downto
          0);
160     get_new_value : out std_logic;
161     not_sclk      : in std_logic; -- system clock
162     not_MinF      : in std_logic; -- Mark start of minor frame,
163     not_MajF      : in std_logic; -- not used
164     not_gate      : in std_logic; -- trigger value transfer
165     FIFO_usedw    : in std_logic_vector(fifo_count_bit_width-1
          downto 0);
166     data          : out std_logic -- data output
167           );
168 end component;
169
170
```

```
171
172
173
174
175
176
177
178 -- Whatchdog
179
180 component watchdog is
181   port (  clk        : in std_logic;
182           not_MinF   : in std_logic;
183           not_MajF   : in std_logic;
184           reset      : out std_logic
185           --FIFO_reset   : out std_logic
186       );
187 end component;
188
189
190 begin
191 FIFO_read_req <= get_new_value;
192 adc_clk       <= MCLK;
193 nFSI          <= '1';
194 Data          <= data_out;
195 test      <= x"85b2";
196
197 Inst_read: read
198   generic map(  number_of_adc => number_of_adc,
199                 regwidth      => regwidth)
200   port map(     clk           => sys_clk,
201                 rst           => sys_rst,
202                 nFSO          => nFSO,
203                 SDO           => SDO,
204                 Rcontr        => read_enable,
205                 SCO           => SCO,
206                 data_vec      => read_in_vec,
207                 datavalid     => FIFO_write_req);
208
209 Inst_interface: interface_sonderakett
210   generic map(  data_out_width       => data_out_width,
211                 fifo_count_bit_width  => fifo_count_bit_width)
212
213   port map(     clk           => sys_clk,
214                 rst           => sys_rst,
215                 value_to_send => value_to_send,
216                 get_new_value => get_new_value,
217                 not_sclk      => not_sclk,
218                 not_MinF      => not_MinF,
219                 not_MajF      => not_Majf,
```

```
220                 not_gate       => not_gate,
221                 FIFO_usedw     => usedw,
222                 data           => data_out);
223
224
225 Inst_fifo: Fifo_16bit
226   port map (       clock => sys_clk,
227                    data  => read_in_vec,
228                    rdreq => FIFO_read_req,
229                    sclr  => sys_rst,
230                    wrreq => FIFO_write_req,
231                    empty => FIFO_empty,
232                    full  => FIFO_full,
233                    q     => value_to_send,
234                    usedw => usedw );
235
236 sys_clk <= clk_50MHz;
237
238 Inst_clk_div: clk_div
239   port map  (      inclk0  => clk_50MHz,
240                    c0      => MCLK,
241                    --c1       => aa_clock,
242                    locked  => clk_ok
243         );
244
245
246 Inst_controller: controller
247   generic map ( fifo_count_bit_width  => fifo_count_bit_width,
248                 fifo_word_length      => fifo_word_length)
249   port map  (    clk          => MCLK,
250                  rst          => sys_rst,
251                  FIFO_empty   => FIFO_empty,
252                  FIFO_full    => Fifo_full,
253                  Wcount       => usedw,
254                  nSync        => nSync,
255                  nReset       => nReset_nPwdwn,
256                  read_enable  => read_enable
257
258                  );
259
260 InstWatch: watchdog
261   port map (   clk        => clk_50MHz,
262                not_MinF   => not_MinF,
263                not_Majf   => not_Majf,
264                reset      => sys_rst
265               --FIFO_reset => FIFO_reset
266             );
267 end str;
```

read.vhd

```
1 ----------------------------------------------------
2 --  Name:               adc_ctrl_sm_ver1_ent
3 --  Writer:             Elling Hauge-Iversen
4 --  Date:               20.1.2012
5 --  Ver:                1
6 --  Description:        Statemachine for reading from adc ad7765.
7
8 --  signals:
9 --  clk           : clock
10 --  rst           : reset
11 --  nFSO          : Frame sync out. Frames the dataset from the
      first adc.
12 --  SDO           : Serial data out. Data out from ADC.
13 --  Rcontr        : Read control. Activates read cycle.
14 --  databit       : Data in to shift register.
15 --  shift         : shifting the shiftregister.
16 --  datavalid     : indicating that the data in the shiftregister
      is valid.
17 --  SCO           : serial clock out from ADC
18 --
19 ----------------------------------------------------
20
21 library IEEE;
22 use IEEE.std_logic_1164.all;
23 use IEEE.numeric_std.all;
24
25 entity read is
26   generic (
27     number_of_adc : natural := 6;
28     regwidth      : natural := 32);
29   port (
30     clk           : in std_logic;
31     rst           : in std_logic;
32     nFSO          : in std_logic;
33     SDO           : in std_logic;
34     Rcontr        : in std_logic; -- enable read
35     SCO           : in std_logic;
36
37     data_vec      : out std_logic_vector(15 downto 0);
38     datavalid     : out std_logic;
39     read_ok       : out std_logic
40   );
41
42 end entity;
43
44
45
```

```
46
47
48 architecture rtl of read is
49
50 component counter is
51   generic (
52     countbit  :  NATURAL; -- counting vector width.
53     goal      :  NATURAL;  -- goal of counter.
54     edge      :  NATURAL --  rising (0) or falling (1)
55     );
56
57   port (
58     clk       : in std_logic;
59     rst       : in std_logic;
60     enable    : in std_logic;
61     countrst  : in std_logic;
62     count     : out std_logic;
63     countvec  : out unsigned(countbit - 1 downto 0)
64     );
65
66 end component;
67
68 component FSO_DET is
69
70   port (
71     clk       : in std_logic;
72     rst       : in std_logic;
73     sigin     : in std_logic; -- clock, reset and signal in
74     sigout    : out std_logic -- signal out
75     );
76
77 end component FSO_DET;
78
79
80
81 signal Rcount         : std_logic;
82 signal ADCcount       : std_logic;
83 signal Renable        : std_logic;
84 signal ADCenable      : std_logic;
85 signal countrst       : std_logic;
86 signal sco_det        : std_logic;
87 signal sco_edge_det   : std_logic;
88 signal FSO_D          : std_logic;
89 signal shift          : std_logic;
90 signal shift_vector   : std_logic_vector(regwidth-1 downto 0);
91 signal databit        : std_logic;
92 signal datavalid_i    : std_logic;
93
94 type statetype is ( init, waiting, holding, reading, shifting,
```

```
    dataval,u1, u2);
95 signal state                : statetype;
96
97
98 begin
99 --countrst <= '0';
100 -- READ COUNTER
    ------------------------------------------------------------
101 --
102 -- counting databits coming in from the adc. When LSB comes in,
    count goes high.
103 --
104 INST_Rcount: counter
105   generic map(
106     countbit  => 7,
107     goal      => regwidth,
108     edge      => 1)
109   port map(
110     clk       => clk,
111     rst       => rst,
112     enable    => Renable,
113     count     => Rcount,
114     countrst  => countrst
115     );
116
117
118
119 -- ADC COUNTER
    ------------------------------------------------------------
120 --
121 -- counting data from adcs in a daisy chain. when reading from
    the last adc, count goes high.
122 INST_ADCcount: counter
123   generic map(
124     countbit  => 7,
125     goal      => number_of_adc,    --- number of ADCs
126     edge      => 1)    -- rising edge
127   port map(
128     clk       => clk,
129     rst       => rst,
130     enable    => ADCenable,
131     count     => ADCcount,
132     countrst  => countrst
133     );
134
135
136
137
138
```

```
139
140
141 INST_FSO_DET: FSO_DET
142   port map(
143     clk      => clk,
144     rst      => rst,
145     sigin    => nFSO,
146     sigout   => FSO_D
147     );
148
149
150 -- Detect fallin gedge on SCO
    ------------------------------------------------
151
152 sco_d: process(rst, clk) is
153 begin
154   if rst = '1' then
155     sco_det <= '0';
156   elsif rising_edge(clk) then
157     sco_det <= SCO;
158     if SCO = '0' and sco_det = '1' then
159       sco_edge_det <= '1';
160     else
161       sco_edge_det <= '0';
162     end if;
163   end if;
164 end process;
165
166 -- STATE MACHINE
    ------------------------------------------------------------
167 --
168 -- State machine for controlling reading of datastream and
    counters.
169
170
171
172 com: process(rst, clk)
173 begin
174   -- end default values
175   if rst = '1' then
176     state <= init;
177   elsif rising_edge(clk) then
178
179       -- DEFAULT VALUES
180     datavalid_i <= '0';
181     --databit <= 'Z';
182     shift <= '0';
183     ADCenable <= '0';
184     Renable <= '0';
```

```
185     countrst <= '0';
186     read_ok <= '0';
187
188     case state is
189       -- INIT STATE
190       when init =>
191         state <= waiting;
192
193       --WAITING STATE
194       when waiting =>
195         if Rcontr = '1' and FSO_D = '1' then
196           state <= holding;
197         else
198           state <= waiting;
199         end if;
200
201       -- HOLDING STATE
202       when holding =>
203         if SCO_EDGE_DET = '1' then
204           state <= reading;
205         else
206           state <= holding;
207         end if;
208
209
210
211       -- READING STATE
212       when reading =>
213         databit <= SDO;
214         state <= shifting;
215
216
217       -- DATAVAL STATE
218       when dataval =>
219 --          if shift_vector(7) = '0' then
220 --            datavalid_i <= '0';
221 --            countrst <= '1';
222 --            state <= waiting;
223 --          else
224             datavalid_i <= '1';
225             ADCenable <= '1';
226             read_ok <= '1';
227
228               if ADCcount = '1' then
229                 state <= waiting;
230               else
231                 state <= holding;
232
233               end if;
```

```
234         --end if;
235
236       -- SHIFTING STATE
237       when shifting =>
238         shift <= '1';
239         Renable <= '1';
240         if Rcount = '1' then
241           state <= dataval;
242         else
243           state <= holding;
244         end if;
245
246       when others =>
247         state <= init;
248
249     end case;
250
251   end if;
252 end process;
253
254
255 shifter: process(rst, clk, shift)
256 begin
257   if rst = '1' then
258     shift_vector <= (others => '0');
259   elsif rising_edge(clk) then
260     if shift = '1' then
261
262       for i in 0 to regwidth-1 loop
263
264         if i = 0 then
265           shift_vector(i) <= databit;
266
267
268         elsif (i>0) then
269           shift_vector(i) <= shift_vector(i-1);
270
271         end if;
272       end loop;
273     end if;
274   end if;
275 end process;
276
277 datavalid <= datavalid_i;
278 data_vec <= shift_vector(31 downto 16) when datavalid_i = '1'
      else "ZZZZZZZZZZZZZZZZ";
279 end rtl;
```

counter.vhd

```
1  ----------------------------------------------------------------
2  --  Name:               counter_ver2_ent
3  --  Writer:             Elling Hauge-Iversen
4  --  Date:               25.1.2012
5  --  Ver:                2
6  --  Description:        Takes in clock reset and enable signal,
7  --                      and returns an array and a signal.
8  --                      Each time enable is goes high, an counter
9  --                      array is incrementet with 1. when counter
10 --                      reaches predefined goal, signal
11 --                      count goes high.
12 ----------------------------------------------------------------
13
14
15
16
17
18 library IEEE;
19 use IEEE.std_logic_1164.all;
20 use IEEE.numeric_std.all;
21
22 entity counter is
23   generic ( countbit   :  NATURAL; -- counting vector width.
24             goal       :  NATURAL;  -- goal of counter.
25             edge       :  NATURAL --  rising (0) or falling (1)
26           );
27
28   port (    clk        : in std_logic;
29             rst        : in std_logic;
30             enable     : in std_logic;
31             countrst   : in std_logic;
32             count      : out std_logic;
33             countvec   : out unsigned(countbit - 1 downto 0)
34         );
35
36 end entity;
37
38
39
40
41
42
43
44
45
46
47
```

```
48
49
50
51
52
53
54 architecture rtl of counter is
55
56 signal countvec_i : unsigned(countbit - 1 downto 0); -- internal
       counter
57
58
59
60 begin
61
62
63 Edgerising:if edge = 0 generate
64 COUN: process(rst, enable, clk)
65 begin
66   if rst = '1' then --or countrst = '1' then
67     countvec_i <= (others => '0');     -- reset counter
68
69   elsif edge = 0 then     -- trig on rising edge
70     if rising_edge(clk) then
71       if countrst = '1' then
72          countvec_i <= (others => '0');
73       elsif enable = '1' then
74
75
76
77         if to_integer(countvec_i) = (goal -1)   then      -- if
              counter has reached goal,
78
79           countvec_i <= (others => '0'); --counter reset to 0
80         else
81
82           countvec_i <= countvec_i + 1;    -- counter incremented.
83         end if;
84       end if;
85
86     end if;
87   end if;
88 end process;
89 end generate;
90
91
92
93
94
```

```
95
96
97
98
99
100
101 edgefalling: if edge = 1 generate
102 COUN: process(rst, enable, clk)
103 begin
104     if rst = '1' then --or countrst = '1' then
105       countvec_i <= (others => '0');     -- reset counter
106     elsif edge = 1 then  -- trig on falling edge
107       if falling_edge(clk) then
108         if countrst = '1' then
109           countvec_i <= (others => '0');
110         elsif enable = '1' then
111
112                     -- if enable is active
113
114           if to_integer(countvec_i) = (goal -1)  then      -- if
                counter has reached goal,
115
116             countvec_i <= (others => '0'); --counter reset to 0
117           else
118
119             countvec_i <= countvec_i + 1;   -- counter
                  incremented.
120           end if;
121         end if;
122       end if;
123
124     end if;
125
126 end process;
127 end generate;
128
129
130 count <= '1' when (to_integer(countvec_i) = goal - 1) else '0';
131 countvec <= countvec_i; -- countvec is countvec_i
132 end rtl;
```

FSO_DET.vhd

```
1 ----------------------------------------------------------
2 --  Name:               FSO_DET_ent
3 --  Writer:             Elling Hauge-Iversen
4 --  Date:               18.1.2012
5 --  Ver:                1
6 --  Description:        Detects falling edge.
7 ----------------------------------------------------------
8
9 library IEEE;
10 use IEEE.std_logic_1164.all;
11 use IEEE.numeric_std.all;
12
13 entity FSO_DET is
14   port ( clk              : in std_logic;
15          rst              : in std_logic;
16          sigin            : in std_logic; -- signal in
17          sigout           : out std_logic -- signal out
18        );
19
20 end entity FSO_DET;
21
22 architecture rtl of FSO_DET is
23 signal sig1      : std_logic;
24 signal sigo1     : std_logic;
25 signal sigo2     : std_logic; -- two internal signals
26
27 begin
28 BITCHECK:  process(rst, clk)
29   begin
30   if rst = '1' then
31     sig1 <= '0';
32       -- if reset, both signals to '0'
33   elsif rising_edge(clk) then
34     sig1 <= sigin;  -- on rising clock edge sig1 equals sigin
35     sigo2 <= sigo1; -- signal out, so that sigout can be 1 more
          than 1 cycle
36     -- comparing sig1 and sig2, if sig1 = 0 and sig2 = 1
37     if (sig1 = '1' and sigin = '0') then
38       sigo1 <= '1';   -- if falling edge, then '1'
39     else
40       sigo1 <= '0';   -- if not sigout is 0;
41     end if;
42   end if;
43 end process;
44 sigout <= sigo1 or sigo2; -- '1' if sigo1 or sigo2 is 1
45 end rtl;
```

controller.vhd

```
1 ------------------------------------------------------------
2 --  Name:              controller
3 --  Writer:            Elling Hauge-Iversen
4 --  Date:              18.1.2012
5 --  Ver:               1
6 --  Description:       Synkronisering av ADC
7 --                     Kontrollere inn- og utlesing
8 --                     av FIFO
9 ------------------------------------------------------------
10
11
12 library IEEE;
13 use IEEE.std_logic_1164.all;
14 use IEEE.numeric_std.all;
15
16 entity controller is
17   generic (
18     fifo_count_bit_width : natural := 2;
19     fifo_word_length     : natural := 200);
20
21   port (
22         clk          : in std_logic;
23         rst          : in std_logic;
24         Fifo_empty   : in std_logic;
25         Fifo_full    : in std_logic;
26         Wcount       : in std_logic_vector(fifo_count_bit_width-1
               downto 0);
27         nsync        : out std_logic;
28         nReset       : out std_logic;
29         read_enable : out std_logic
30         );
31 end entity;
32
33 architecture rtl of controller is
34
35 component counter is
36   generic ( countbit  :  NATURAL; -- counting vector width.
37             goal      :  NATURAL;  -- goal of counter.
38             edge      :  NATURAL --  rising (0) or falling (1)
39           );
40
41   port (    clk        : in std_logic;
42             rst        : in std_logic;
43             enable     : in std_logic;
44             countrst   : in std_logic;
45             count      : out std_logic;
46             countvec   : out unsigned(countbit - 1 downto 0)
```

```
47          );
48 end component;
49
50 constant countbit : NATURAL := 3;
51 constant goal     : NATURAL := 4;
52 constant edge     : NATURAL := 0;
53
54 type controller_state is (Reset, Sync, yes_readin, no_readin);
55 signal c_state      : controller_state;
56 signal r_enable     : std_logic;
57 signal r_countrst   : std_logic;
58 signal r_count      : std_logic;
59 signal s_enable     : std_logic;
60 signal s_countrst   : std_logic;
61 signal s_count      : std_logic;
62 signal r_countvec   : unsigned(countbit-1 downto 0);
63 signal s_countvec   : unsigned(countbit-1 downto 0);
64
65
66 begin
67
68 Inst_r_counter: counter
69   generic map ( countbit => countbit,
70           goal      => goal,
71           edge      => edge)
72   port map (    clk => clk,
73           rst       => rst,
74           enable    => r_enable,
75           countrst  => r_countrst,
76           count     => r_count,
77           countvec  => r_countvec
78           );
79
80 Inst_s_counter: counter
81   generic map (
82           countbit  => countbit,
83           goal      => goal,
84           edge      => edge)
85   port map (
86           clk       => clk,
87           rst       => rst,
88           enable    => s_enable,
89           countrst  => s_countrst,
90           countvec  => s_countvec,
91           count     => s_count
92           );
93
94 Proc_controll_state: process(clk, rst)
95 begin
```

```
 96   if rst = '1' then
 97     c_state <= reset;
 98   elsif rising_edge(clk) then
 99     nsync <= '1';
100     nReset <= '1';
101     read_enable <= '0';
102     r_enable <= '0';
103     s_enable <= '0';
104     r_countrst <= '0';
105     s_countrst <= '0';
106
107     case c_state is
108       when Reset =>
109         nreset <= '0';
110         --read_enable <= '0';
111         --r_countrst <= '1';
112         --s_countrst <= '1';
113         if r_count = '1' then
114           c_state <= sync;
115         else
116           r_enable <= '1';
117           c_state <= reset;
118         end if;
119       when sync =>
120         nsync <= '0';
121         if s_count = '1' then
122           c_state <= yes_readin;
123         else
124           s_enable <= '1';
125           c_state <= sync;
126         end if;
127       when yes_readin =>
128         read_enable <= '1';
129         if to_integer(unsigned(Wcount)) > (fifo_word_length-10)
              then
130           c_state <= no_readin;
131         else
132           c_state <= yes_readin;
133         end if;
134       when no_readin =>
135         --read_enable <= '0';
136         if Fifo_empty = '1' then
137           c_state <= yes_readin;
138         else
139           c_state <= no_readin;
140         end if;
141     end case;
142   end if;
143 end process;
```

```
144 end architecture;
```

interface_sonderakett.vhd

```
1  ----------------------------------------------------------
2  --  Name:               interface
3  --  Writer:             Elling Hauge-Iversen
4  --  Date:               22.11.2012
5  --  Ver:                1
6  --  Description:        Styre utlesning til encoder
7
8  ----------------------------------------------------------
9
10
11
12 library IEEE;
13 use IEEE.std_logic_1164.all;
14 use IEEE.numeric_std.all;
15 use ieee.std_logic_unsigned.all;
16
17
18
19
20
21
22
23 entity interface_sonderakett is
24   generic (    data_out_width  : integer := 16;
25                First_valid_word_in_frame : integer := 1;
26                Valid_word_increment    : integer    := 2;
27                fifo_count_bit_width : integer);
28
29   port  ( clk         : in std_logic; -- internal clock
30           rst         : in std_logic; -- reset
31           value_to_send    : in std_logic_vector(data_out_width-1
                downto 0);
32           get_new_value : out std_logic;
33           not_sclk        : in std_logic; -- system clock
34           not_MinF        : in std_logic; --  minor frame
35           not_MajF        : in std_logic; -- major frame
36           not_gate        : in std_logic; -- trigger value transfer
37           FIFO_usedw      : in std_logic_vector(
                fifo_count_bit_width-1 downto 0);
38           data            : out std_logic -- data output
39           );
40 end entity;
41
42
43
44
```

```
45
46
47
48 architecture rtl of interface_sonderakett is
49
50 ------------------------------------------------------------
51 --                    COMPONENT DECLARATIONS
52 ------------------------------------------------------------
53 component Shiftreg is
54   generic (
55     width : integer := 96);
56
57
58   port (
59     clk      : in  std_logic;
60     DataIn   : in  std_logic_vector(width-1 downto 0);
61     load     : in  std_logic;
62     shift_en : in  std_logic;
63     reset    : in  std_logic;
64     DataOut  : out std_logic);
65
66 end component;
67
68 ------------------------------------------------------------
69 --                    SIGNALS
70 ------------------------------------------------------------
71
72 -- Signals from TM encoder
73 signal sclk              : std_logic;
74 signal minf              : std_logic;
75 signal gate              : std_logic;
76 signal majf              : std_logic;
77 signal spare             : std_logic;
78
79 -- Internal reset signals
80 signal reset             : std_logic;
81 signal reset_sync        : std_logic := '0';    -- Synchronize to
       TM-encoder
82 -- Internal TM counters
83 signal bit_cnt           : std_logic_vector(2 downto 0);
84 signal WordStrobe        : std_logic;
85 signal word_cnt          : std_logic_vector(7 downto 0);
86 --signal frame_cnt         : std_logic_vector(1 downto 0);
87 signal value_out_cnt     : std_logic_vector(2 downto 0);
88
89 signal gate_delayed, gate_delayed1     : std_logic;
                            -- To not loose MSB in TM
90 signal DataSample        : std_logic_vector(95 downto 0);   --
       Data to TM Shift reg.
```

```
91 signal load_sample       : std_logic;
92 signal load_sample_del   : std_logic;                    --
       Enable/disable TM shift reg
93
94 -- Type definitions
95 type statetype1 is (SyncCheck, IgnoreSt);
96 signal sync_state : statetype1;
97 type statetype4 is (init, waiting, loadvalues, wait_load1,
       wait_load0 );
98 signal fetchstate : statetype4;
99 signal fifo_count: unsigned(3 downto 0);
100 signal Fifo_rdreq : std_logic;
101 signal data_vec_out : std_logic_vector(95 downto 0);
102
103
104
105
106 begin
107
108 ------------------------------------------------------------
109 --                        PORT MAPPING
110 ------------------------------------------------------------
111
112 -- Data transfer shift register
113 Com_SR: shiftreg
114
115 port map (   clk => sclk,
116              Datain => data_vec_out,
117              load => load_sample_del,
118              shift_en => gate_delayed,
119              reset => rst,
120              dataout => data);
121
122
123
124 ------------------------------------------------------------
125
126 -- Invert again the signals inverted in the optocouplers
127 sclk <= not(not_sclk);
128 minf <= not(not_minf);
129 majf <= not(not_majf);
130 gate <= not(not_gate);
131
132
133
134
135
136
137
```

```
138
139
140
141 ------------------------------------------------------------
142 --                        WORD STROBE
143 ------------------------------------------------------------
144
145 BITCOUNTER:
146 process(sclk, reset_sync)
147 begin
148   if reset_sync = '1' then              -- asyncron
149     bit_cnt <= (others => '0');
150   elsif falling_edge(sclk) then
151     bit_cnt <= bit_cnt + 1;
152   end if;
153 end process;
154
155 WORDSYNC:
156 process(sclk)
157 begin
158   if rising_edge(sclk) then
159     if (bit_cnt = 7) then
160       WordStrobe <= '1';
161     else
162       WordStrobe <= '0';
163     end if;
164   end if;
165 end process;
166
167 ------------------------------------------------------------
168 --                        WORD COUNTER
169 ------------------------------------------------------------
170
171 -- 144 words per frame
172
173 WORDCOUNTER:
174 process(sclk, reset_sync)
175 begin
176   if (reset_sync = '1') then
177     word_cnt <= (others => '0');
178   elsif falling_edge(sclk) then
179     if (WordStrobe = '1') then
180       if (word_cnt < 143) then
181         word_cnt <= word_cnt + 1;
182       else
183         word_cnt <= (others => '0');
184       end if;
185     end if;
186   end if;
```

```
187 end process;
188
189
190 ------------------------------------------------------------
191 --                        FSM  SYNCRONIZERS
192 ------------------------------------------------------------
193 -- Synchronize / reset bit & word counter to minf from PCM
       encoder
194
195 -- Synchronize / reset bit & word counter to minf and majf from
       encoder
196 -- Creates a short reset pulse. For the reset pulse to be short
       compared to a
197 -- clock period from the system (encoder) clock, the PCB
       oscillator clock
198 -- frequency must be considerably higher that the system clock
       frequency. A ratio
199 -- where the PCB oscillator is 6 timer higher than the system
       clock has been tested
200 -- and verified to work without glitches.
201
202
203 FSM_MINF_MAJF_SYNC:
204 process(clk)
205 begin
206   if rising_edge(clk) then
207     case sync_state is
208       when SyncCheck =>
209         --if (minf = '1' or majf = '1') then  "majf not in use"
210         if (minf = '1') then
211           reset_sync <= '1';
212           sync_state <= IgnoreSt;
213         else
214           reset_sync <= '0';
215           sync_state <= SyncCheck;
216         end if;
217
218       when IgnoreSt =>
219         reset_sync <= '0';
220         --if (minf = '0' and majf = '0') then  "majf not in use"
221         if (minf = '0') then
222           sync_state <= SyncCheck;
223         else
224           sync_state <= IgnoreSt;
225         end if;
226     end case;
227   end if;
228 end process;
229
```

```
230
231
232
233
234 ------------------------------------------------------------
235 --                          GATE DELAY
236 ------------------------------------------------------------
237 -- Delay of the gate signal, to avoid losing msb
238
239 GATEDELAY:
240 process(sclk)
241 begin
242   if rising_edge(sclk) then
243     if(gate = '1') then
244       gate_delayed <= '1';
245     else
246       gate_delayed <= '0';
247     end if;
248    --gate_delayed1 <= gate_delayed;
249   end if;
250 end process;
251
252
253
254
255
256
257 ------------------------------------------------------------
258 --                          Word_load
259 ------------------------------------------------------------
260
261
262 DATA_TRANSFER:
263 process(sclk)
264 begin
265   if rising_edge(sclk) then
266     load_sample <= '0';
267
268      -- CH1 --> CH8 into 128-bit SR
269     if ((word_cnt = 1 or
270          word_cnt = 25 or
271          word_cnt = 49 or
272          word_cnt = 73 or
273          word_cnt = 97 or
274          word_cnt = 121
275        ) and bit_cnt = 6) then
276       data_vec_out <= DataSample;
277       load_sample <= '1';
278     end if;
```

```
279   end if;
280 end process;
281
282
283
284 -- Delay the load signal to SR with one half clock periode,
285 -- because data are transmitted from the shift register on the
286 -- rising edge, the same flank as the data transfer process uses.
287 LOAD_DELAY:
288 process(sclk)
289 begin
290   if falling_edge(sclk) then
291     if (load_sample = '1') then
292       load_sample_del <= '1';
293     else
294       load_sample_del <= '0';
295     end if;
296   end if;
297 end process;
298
299
300
301
302
303
304
305 ------------------------------------------------------------
306 -- Load out samples from FIFO
307 ------------------------------------------------------------
308
309 FIFOFETCH: process(clk)
310 begin
311   if falling_edge(clk) then
312     get_new_value <= '0';
313     if reset_sync = '1' then
314       fetchstate <= init;
315
316     else
317
318       case fetchstate is
319
320         when init =>
321           fetchstate <= waiting;
322           fifo_count <= (others => '0');
323
324         when waiting =>
325           if FIFO_usedw < 6 then -- wait to at least 6 words in
                FIFO
326             fetchstate <= waiting;
```

```
327             else
328               fetchstate <= loadvalues;
329             end if;
330
331           when loadvalues =>
332             get_new_value <= '1';
333             if fifo_count = 0 then
334               DataSample(95 downto 80) <= value_to_send;
335             elsif fifo_count = 1 then
336               DataSample(79 downto 64) <= value_to_send;
337             elsif fifo_count = 2 then
338               DataSample(63 downto 48) <= value_to_send;
339             elsif fifo_count = 3 then
340               DataSample(47 downto 32) <= value_to_send;
341             elsif fifo_count = 4 then
342               DataSample(31 downto 16) <= value_to_send;
343             elsif fifo_count = 5 then
344               DataSample(15 downto 0) <= value_to_send;
345             end if;
346
347             if fifo_count = 5 then
348               fifo_count <= (others => '0');
349               fetchstate <= wait_load1;
350             else
351               fifo_count <= fifo_count+1;
352               fetchstate <= loadvalues;
353             end if;
354
355           when wait_load1 =>
356             if load_sample = '1' then
357               fetchstate <= wait_load0;
358             else
359               fetchstate <= wait_load1;
360             end if;
361
362           when wait_load0 =>
363             if load_sample = '0' then
364               fetchstate <= waiting;
365             else
366               fetchstate <= wait_load0;
367             end if;
368         end case;
369       end if;
370     end if;
371 end process;
372
373
374
375 end rtl;
```

shiftreg.vhd

```
1  ----------------------------------------------------------------
2  --  Name:               shiftreg
3  --  Writer:             Elling Hauge-Iversen
4  --  Date:               22.11.2012
5  --  Ver:                1
6  --  Description:        Shiftregister. skifter
7  --                      dataverdiene ut paa
8  --                      datalinjen naar gate
9  --                      er hoey.
10
11 ----------------------------------------------------------------
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.std_logic_arith.all;
15 use ieee.std_logic_unsigned.all;
16
17
18
19
20 entity shiftreg is
21   generic (
22     width : integer := 96);
23
24
25   port (
26     clk      : in  std_logic;
27     DataIn   : in  std_logic_vector(width-1 downto 0);
28     load     : in  std_logic;
29     shift_en : in  std_logic;
30     reset    : in  std_logic;
31     DataOut  : out std_logic);
32
33 end shiftreg;
34
35
36
37
38
39
40
41
42
43
44
45
46
47
```

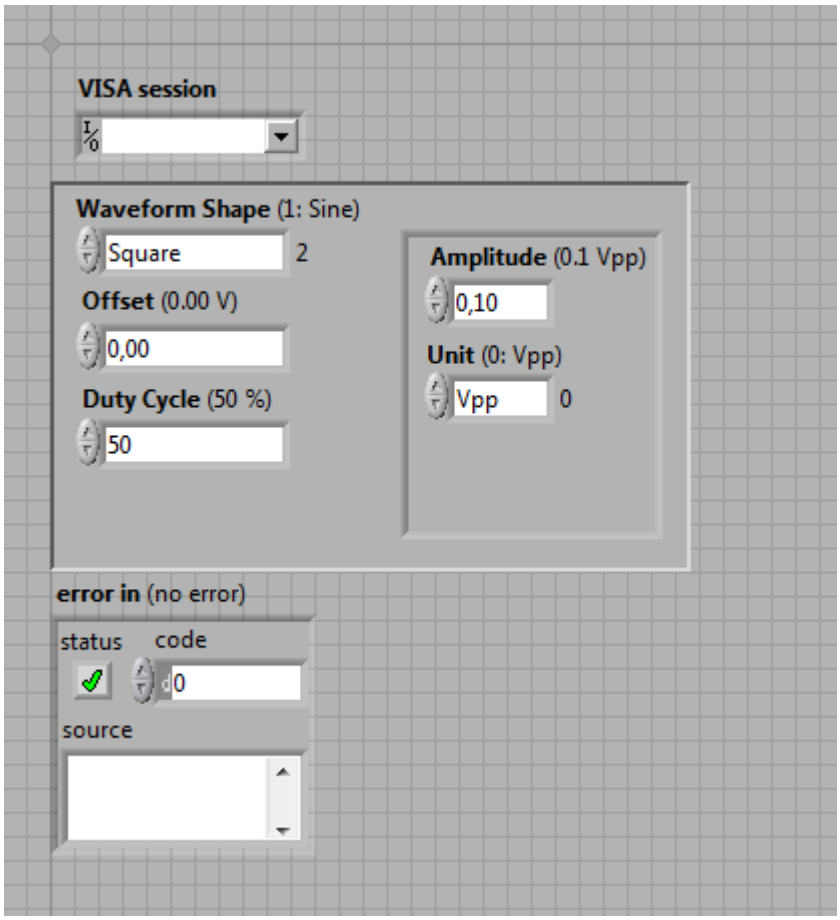
```
48
49
50 architecture SR_par2ser_arch of shiftreg is
51
52   signal data_int : std_logic_vector(width-1 downto 0);
53
54
55 begin
56
57   SHIFT_REG: process (clk, reset)
58   begin
59
60     if reset = '1' then                    -- asynchronous reset
61       data_int <= (others => '0');
62
63     elsif rising_edge(clk) then
64       if (load = '1') then
65         data_int <= DataIn;
66
67       elsif (shift_en = '1') then
68         for i in width-2 downto 0 loop
69           data_int(i+1) <= data_int(i);
70         end loop;
71       end if;
72
73     end if;
74   end process SHIFT_REG;
75
76   DataOut <= data_int(width-1);
77 --DataOut <= clk;
78
79
80
81 end SR_par2ser_arch;
```

watchdog.vhd

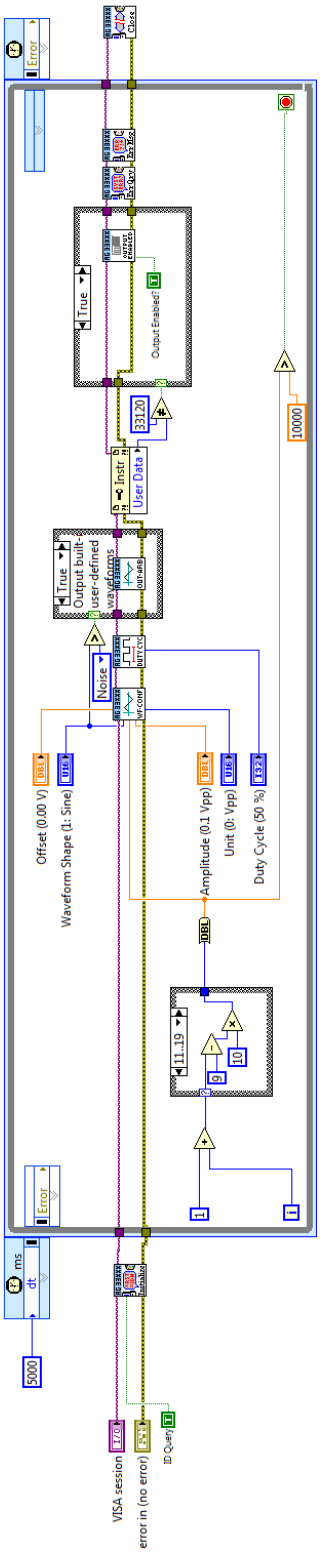
```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4
5 entity watchdog is
6   port (  clk      :in std_logic;
7        not_MinF   : in std_logic;
8        not_MajF   : in std_logic;
9        reset      : out std_logic
10       --FIFO_reset   : out std_logic
11       );
12 end entity;
13
14 architecture rtl of watchdog is
15
16 --type watchdogtype is (init, res, waiting);
17 --signal w_state : watchdogtype := init;
18 signal r_count : unsigned(7 downto 0) := (others => '0');
19
20
21 begin
22
23
24 proc_reset_count: process(not_majf)
25 begin
26   if falling_edge(not_majF) then
27        r_count <= r_count +1;
28   end if;
29 end process;
30
31 --FIFO_reset <= '0';
32
33 Proc_FIFO_reset: process(clk)
34 begin
35   if rising_edge(clk) then
36     if not_MajF = '0' and r_count = x"0000" then
37       reset <= '1';
38     else
39       reset <= '0';
40
41
42     end if;
43   end if;
44 end process;
45
46 end rtl;
```

Tillegg G

Labview-filer



Figur G.1: Labview testprogram frontpanel.



Figur G.2: Labview testprogram blokkskjema.